**Работа с базами данных SQLite**

**Подключение к базе данных SQLite**

**Последнее обновление: 05.03.2017**

В Android имеется встроенная поддержка одной из распространенных систем управления базами данных - SQLite. Для этого в пакете **android.database.sqlite** определен набор классов, которые позволяют работать с базами данных SQLite. И каждое приложение может создать свою базу данных.

Чтобы использовать SQLite в Android, надо создать базу данных с помощью выражение на языке SQL. После этого база данных будет храниться в каталоге приложения по пути:

|  |  |
| --- | --- |
| 1 | DATA/data/[Название\_приложения]/databases/[Название\_файла\_базы\_данных] |

ОС Android по умолчанию уже содержит ряд встроенных бад SQLite, которые используются стандартными программами - для списка контактов, для хранения фотографий с камеры, музыкальных альбомов и т.д.

Основную функциональность по работе с базами данных предоставляет пакет **android.database**. Функциональность непосредственно для работы с SQLite находится в пакете **android.database.sqlite**.

База данных в SQLite представлена классом **android.database.sqlite.SQLiteDatabase**. Он позволяет выполнять запросы к бд, выполнять с ней различные манипуляции.

Класс **android.database.sqlite.SQLiteCursor** предоставляет запрос и позволяет возвращать набор строк, которые соответствуют этому запросу.

Класс **android.database.sqlite.SQLiteQueryBuilder** позволяет создавать SQL-запросы.

Сами sql-выражения представлены классом **android.database.sqlite.SQLiteStatement**, которые позволяют с помощью плейсхолдеров вставлять в выражения динамические данные.

Класс **android.database.sqlite.SQLiteOpenHelper** позволяет создать базу данных со всеми таблицами, если их еще не существует.

В SQLite применяется следующая система типов данных:

* **INTEGER**: представляет целое число, аналог типу int в java
* **REAL**: представляет число с плавающей точкой, аналог float и double в java
* **TEXT**: представляет набор символов, аналог String и char в java
* **BLOB**: представляет массив бинарных данных, например, изображение, аналог типу int в java

Сохраняемые данные должны представлять соответствующие типы в java.

**Создание и открытие базы данных**

Для создания или открытия новой базы данных из кода Activity в Android мы можем вызвать метод **openOrCreateDatabase()**. Этот метод может принимать три параметра:

* название для базы данных
* числовое значение, которое определяет режим работы (как правило, в виде константы MODE\_PRIVATE)
* необязательный параметр в виде объекта SQLiteDatabase.CursorFactory, который представляет фабрику создания курсора для работы с бд

Например, создание базы данных app.db:

|  |  |
| --- | --- |
| 1 | SQLiteDatabase db = getBaseContext().openOrCreateDatabase("app.db", MODE\_PRIVATE, null); |

Для выполнения запроса к базе данных можно использовать метод **execSQL** класса SQLiteDatabase. В этот метод передается SQL-выражение. Например, создание в базе данных таблицы users:

|  |  |
| --- | --- |
| 1  2 | SQLiteDatabase db = getBaseContext().openOrCreateDatabase("app.db", MODE\_PRIVATE, null);  db.execSQL("CREATE TABLE IF NOT EXISTS users (name TEXT, age INTEGER)"); |

Если нам надо не просто выполнить выражение, но и получить из бд какие-либо данные, то используется метод **rawQuery()**. Этот метод в качестве параметра принимает SQL-выражение, а также набор значений для выражения sql. Например, получение всех объектов из базы данных:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | SQLiteDatabase db = getBaseContext().openOrCreateDatabase("app.db", MODE\_PRIVATE, null);  db.execSQL("CREATE TABLE IF NOT EXISTS users (name TEXT, age INTEGER)");  Cursor query = db.rawQuery("SELECT \* FROM users;", null);  if(query.moveToFirst()){        String name = query.getString(0);      int age = query.getInt(1);  } |

Метод db.rawQuery() возвращает объект Cursor, с помощью которого мы можем извлечь полученные данные.

Возможна ситуация, когда в базе данных не будет объектов, и для этого методом query.moveToFirst() пытаемся переместиться к первому объекту, полученному из бд. Если этот метод возвратит значение false, значит запрос не получил никаких данных из бд.

Теперь для работы с базой данных сделаем простейшее приложение. Для этого создадим новый проект.

В файле **activity\_main.xml** определим простейший графический интерфейс:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <?xml version="1.0" encoding="utf-8"?>  <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"      android:id="@+id/activity\_main"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent"      android:padding="16dp"      android:orientation="vertical">      <Button          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          android:text="Click"          android:onClick="onClick"/>      <TextView          android:id="@+id/textView"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          android:textSize="18sp" />  </LinearLayout> |

А в классе **MainActivity** определим взаимодействие с базой данных:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | package com.example.eugene.sqliteapp;    import android.database.Cursor;  import android.database.sqlite.SQLiteDatabase;  import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.view.View;  import android.widget.TextView;    public class MainActivity extends AppCompatActivity {        @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          setContentView(R.layout.activity\_main);      }        public void onClick(View view){          SQLiteDatabase db = getBaseContext().openOrCreateDatabase("app.db", MODE\_PRIVATE, null);          db.execSQL("CREATE TABLE IF NOT EXISTS users (name TEXT, age INTEGER)");          db.execSQL("INSERT INTO users VALUES ('Tom Smith', 23);");          db.execSQL("INSERT INTO users VALUES ('John Dow', 31);");            Cursor query = db.rawQuery("SELECT \* FROM users;", null);          TextView textView = (TextView) findViewById(R.id.textView);          if(query.moveToFirst()){              do{                  String name = query.getString(0);                  int age = query.getInt(1);                  textView.append("Name: " + name + " Age: " + age + "\n");              }              while(query.moveToNext());          }          query.close();          db.close();      }  } |

По нажатию на кнопку здесь вначале создается в базе данных app.db новая таблица users, а затем в нее добавляются два объекта в базу данных с помощью SQL-выражения INSERT.

Далее с помощью выражения SELECT получаем всех добавленных пользователей из базы данных в виде курсора Cursor.

Вызовом query.moveToFirst() перемещаемся в курсоре к первому объекту, и так как у нас может быть больше одного объекта, то проходим по всему курсору в цикле do...while.

Для получения данных из курсора применяются методы query.getString(0) и query.getInt(1). В скобках в методы передается номер столбца, из которого мы получаем данные. Например, выше мы добавили вначале имя пользователя в виде строки, а затем возраст в виде числа. Значит, нулевым столбцом будет идти строкое значение, которое получаем с помощью метода getString(), а следующим - первым столбцом идет числовое значение, для которого применяется метод getInt().

После завершения работы с курсором и базой данных мы закрываем все связанные объекты:

|  |  |
| --- | --- |
| 1  2 | query.close();  db.close(); |

Если мы не закроем курсор, то можем столкнуться с проблемой утечки памяти.

И если мы обратимся к приложению, то после нажатия на кнопку в текстовое поле будут выведены добавленные данные:



**SimpleCursorAdapter и получение данных**

**Последнее обновление: 05.03.2017**

В прошлой теме было рассмотрено, как подключаться к базе данных SQLite и выполнять запросы. Теперь пойдем дальше и создадим полностью интерфейс для работы с базой данных.

Итак, создадим новый проект.

Для упрощения работы с базами данных SQLite в Android нередко применяется класс **SQLiteOpenHelper**. Для использования необходимо создать класса-наследник от SQLiteOpenHelper, переопределив как минимум два его метода:

* onCreate(): вызывается при попытке доступа к базе данных, но когда еще эта база данных не создана
* onUpgrade(): вызывается, когда необходимо обновление схемы базы данных. Здесь можно пересоздать ранее созданную базу данных в onCreate(), установив соответствующие правила преобразования от старой бд к новой

Поэтому добавим в проект, в ту же папку, где находится класс MainActivity, новый класс **DatabaseHelper**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | package com.example.eugene.sqliteapp;    import android.database.sqlite.SQLiteOpenHelper;  import android.database.sqlite.SQLiteDatabase;  import android.content.Context;  import android.content.ContentValues;    public class DatabaseHelper extends SQLiteOpenHelper {      private static final String DATABASE\_NAME = "userstore.db"; // название бд      private static final int SCHEMA = 1; // версия базы данных      static final String TABLE = "users"; // название таблицы в бд      // названия столбцов      public static final String COLUMN\_ID = "\_id";      public static final String COLUMN\_NAME = "name";      public static final String COLUMN\_YEAR = "year";        public DatabaseHelper(Context context) {          super(context, DATABASE\_NAME, null, SCHEMA);      }        @Override      public void onCreate(SQLiteDatabase db) {            db.execSQL("CREATE TABLE users (" + COLUMN\_ID                  + " INTEGER PRIMARY KEY AUTOINCREMENT," + COLUMN\_NAME                  + " TEXT, " + COLUMN\_YEAR + " INTEGER);");          // добавление начальных данных          db.execSQL("INSERT INTO "+ TABLE +" (" + COLUMN\_NAME                  + ", " + COLUMN\_YEAR  + ") VALUES ('Том Смит', 1981);");      }      @Override      public void onUpgrade(SQLiteDatabase db, int oldVersion,  int newVersion) {          db.execSQL("DROP TABLE IF EXISTS "+TABLE);          onCreate(db);      }  } |

Если база данных отсутствует или ее версия (которая задается в переменной SCHEMA) выше текущей, то срабатывает метод onCreate().

Для выполнения запросов к базе данных нам потребуется объект **SQLiteDatabase**, который представляет базу данных. Метод onCreate()получает в качестве параметра базу данных приложения.

Для выполнения запросов к SQLite используется метод **execSQL()**. Он принимает sql-выражение CREATE TABLE, которое создает таблицу. Здесь также при необходимости мы можем выполнить и другие запросы, например, добавить какие-либо начальные данные. Так, в данном случае с помощью того же метода и выражения sql INSERT добавляется один объект в таблицу.

В методе onUpgrade() происходит обновление схемы БД. В данном случае для примера использован примитивный поход с удалением предыдущей базы данных с помощью sql-выражения DROP и последующим ее созданием. Но в реальности если вам будет необходимо сохранить данные, этот метод может включать более сложную логику - добавления новых столбцов, удаление ненужных, добавление дополнительных данных и т.д.

Далее определим в файле **activity\_main.xml** следующую разметку:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <?xml version="1.0" encoding="utf-8"?>  <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"      android:orientation="vertical"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent">      <TextView          android:id="@+id/header"          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content"          android:textSize="18dp"/>      <ListView          android:id="@+id/list"          android:layout\_width="match\_parent"          android:layout\_height="match\_parent" />  </LinearLayout> |

Здесь определен список ListView, для отображения полученных данных, с заголовком, который будет выводить число полученных объектов.

И изменим код класса **MainActivity** следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | package com.example.eugene.sqliteapp;    import android.database.Cursor;  import android.database.sqlite.SQLiteDatabase;  import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.view.View;  import android.widget.AdapterView;  import android.widget.ListView;  import android.widget.SimpleCursorAdapter;  import android.widget.TextView;    public class MainActivity extends AppCompatActivity {        ListView userList;      TextView header;      DatabaseHelper databaseHelper;      SQLiteDatabase db;      Cursor userCursor;      SimpleCursorAdapter userAdapter;        @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          setContentView(R.layout.activity\_main);            header = (TextView)findViewById(R.id.header);          userList = (ListView)findViewById(R.id.list);            databaseHelper = new DatabaseHelper(getApplicationContext());      }      @Override      public void onResume() {          super.onResume();          // открываем подключение          db = databaseHelper.getReadableDatabase();            //получаем данные из бд в виде курсора          userCursor =  db.rawQuery("select \* from "+ DatabaseHelper.TABLE, null);          // определяем, какие столбцы из курсора будут выводиться в ListView          String[] headers = new String[] {DatabaseHelper.COLUMN\_NAME, DatabaseHelper.COLUMN\_YEAR};          // создаем адаптер, передаем в него курсор          userAdapter = new SimpleCursorAdapter(this, android.R.layout.two\_line\_list\_item,                  userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2}, 0);          header.setText("Найдено элементов: " + String.valueOf(userCursor.getCount()));          userList.setAdapter(userAdapter);      }        @Override      public void onDestroy(){          super.onDestroy();          // Закрываем подключение и курсор          db.close();          userCursor.close();      }  } |

В методе onCreate() происходит создание объекта SQLiteOpenHelper. Сама инициализация объектов для работы с базой данных происходит в методе onResume(), который срабатывает после метода onCreate().

Чтобы получить объект базы данных, надо использовать метод getReadableDatabase() (получение базы данных для чтения) илиgetWritableDatabase(). Так как в данном случае мы будет только считывать данные из бд, то воспользуемся первым методом:

|  |  |
| --- | --- |
| 1 | db = sqlHelper.getReadableDatabase(); |

**Получение данных и Cursor**

Android предоставляет различные способы для осуществления запросов к объекту SQLiteDatabase. В большинстве случаев мы можем применять метод **rawQuery()**, который принимает два параметра: SQL-выражение SELECT и дополнительный параметр, задающий параметры запроса.

После выполнения запроса rawQuery() возвращает объект **Cursor**, который хранит результат выполнения SQL-запроса:

|  |  |
| --- | --- |
| 1 | userCursor =  db.rawQuery("select \* from "+ DatabaseHelper.TABLE, null); |

Класс Cursor предлагает ряд методов для управления выборкой, в частности:

* getCount(): получает количество извлеченных из базы данных объектов
* Методы moveToFirst() и moveToNext() позволяют переходить к первому и к следующему элементам выборки. Метод isAfterLast()позволяет проверить, достигнут ли конец выборки.
* Методы get\*(columnIndex) (например, getLong(), getString()) позволяют по индексу столбца обратиться к данному столбцу текущей строки

**CursorAdapter**

Дополнительно для управления курсором в Android имеется класс CursorAdapter. Он позволяет адаптировать полученный с помощью курсора набор к отображению в списковых элементах наподобие ListView. Как правило, при работе с курсором используется подкласс CursorAdapter - **SimpleCursorAdapter**. Хотя можно использовать и другие адаптеры, типа ArrayAdapter.

|  |  |
| --- | --- |
| 1  2  3 | userAdapter = new SimpleCursorAdapter(this, android.R.layout.two\_line\_list\_item,                  userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2}, 0);  userList.setAdapter(userAdapter); |

Конструктор класса SimpleCursorAdapter принимает шесть параметров:

1. Первым параметром выступает контекст, с которым ассоциируется адаптер, например, текущая activity
2. Второй параметр - ресурс разметки интерфейса, который будет использоваться для отображения результатов выборки
3. Третий параметр - курсор
4. Четвертый параметр - список столбцов из выборки, которые будут отображаться в разметке интерфейса
5. Пятый параметр - элементы внутри ресурса разметки, которые будут отображать значения столбцов из четвертого параметра
6. Шестой параметр - флаги, задающие поведения адаптера

При использовании CursorAdapter и его подклассов следует учитывать, что выборка курсора должна включать целочисленный столбец с названием **\_id**, который должен быть уникальным для каждого элемента выборки. Значение этого столбца при нажатии на элемент списка затем передается в метод обработки onListItemClick(), благодаря чему мы можем по id идентифицировать нажатый элемент.

В данном случае у нас первый столбец как раз называется "\_id".

После завершения работу курсор должен быть закрыт методом close()

И также надо учитывать, что если мы используем курсор в SimpleCursorAdapter, то мы не можем использовать метод close(), пока не завершим использование SimpleCursorAdapter. Поэтому метод cursor более предпочтительно вызывать в методе onDestroy() фрагмента или activity.

И если мы запустим приложение, то увидим список из одного добавленного элемента:



**Добавление, удаление и обновление данных в SQLite**

**Последнее обновление: 05.03.2017**

Продолжим работу с проектом из прошлой темы, где мы получаем данные. Теперь добавим в него стандартную CRUD-логику (создание, обновление, удаление).

Чтобы не нагромождать форму с главной activity, все остальные действия по работе с данными будут происходить на другом экране. Добавим в проект новый класс activity, который назовем UserActivity:

В файле **activity\_user.xml** определим универсальую форму для добавления/обновления/удаления данных:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | <?xml version="1.0" encoding="utf-8"?>  <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"      android:orientation="vertical"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent"      android:padding="16dp">      <EditText          android:id="@+id/name"          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content"          android:hint="Введите имя"/>      <EditText          android:id="@+id/year"          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content"          android:hint="Введите год рождения"/>      <LinearLayout          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content"          android:orientation="horizontal">            <Button              android:id="@+id/saveButton"              android:layout\_width="0dp"              android:layout\_weight="1"              android:layout\_height="wrap\_content"              android:text="Сохранить"              android:onClick="save"/>          <Button              android:id="@+id/deleteButton"              android:layout\_width="0dp"              android:layout\_weight="1"              android:layout\_height="wrap\_content"              android:text="Удалить"              android:onClick="delete"/>      </LinearLayout>  </LinearLayout> |

И также изменим код **UserActivity**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80 | package com.example.eugene.sqliteapp;    import android.content.ContentValues;  import android.content.Intent;  import android.database.Cursor;  import android.database.sqlite.SQLiteDatabase;  import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.view.View;  import android.widget.Button;  import android.widget.EditText;    public class UserActivity extends AppCompatActivity {        EditText nameBox;      EditText yearBox;      Button delButton;      Button saveButton;        DatabaseHelper sqlHelper;      SQLiteDatabase db;      Cursor userCursor;      long userId=0;      @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          setContentView(R.layout.activity\_user);            nameBox = (EditText) findViewById(R.id.name);          yearBox = (EditText) findViewById(R.id.year);          delButton = (Button) findViewById(R.id.deleteButton);          saveButton = (Button) findViewById(R.id.saveButton);            sqlHelper = new DatabaseHelper(this);          db = sqlHelper.getWritableDatabase();            Bundle extras = getIntent().getExtras();          if (extras != null) {              userId = extras.getLong("id");          }          // если 0, то добавление          if (userId > 0) {              // получаем элемент по id из бд              userCursor = db.rawQuery("select \* from " + DatabaseHelper.TABLE + " where " +                      DatabaseHelper.COLUMN\_ID + "=?", new String[]{String.valueOf(userId)});              userCursor.moveToFirst();              nameBox.setText(userCursor.getString(1));              yearBox.setText(String.valueOf(userCursor.getInt(2)));              userCursor.close();          } else {              // скрываем кнопку удаления              delButton.setVisibility(View.GONE);          }      }        public void save(View view){          ContentValues cv = new ContentValues();          cv.put(DatabaseHelper.COLUMN\_NAME, nameBox.getText().toString());          cv.put(DatabaseHelper.COLUMN\_YEAR, Integer.parseInt(yearBox.getText().toString()));            if (userId > 0) {              db.update(DatabaseHelper.TABLE, cv, DatabaseHelper.COLUMN\_ID + "=" + String.valueOf(userId), null);          } else {              db.insert(DatabaseHelper.TABLE, null, cv);          }          goHome();      }      public void delete(View view){          db.delete(DatabaseHelper.TABLE, "\_id = ?", new String[]{String.valueOf(userId)});          goHome();      }      private void goHome(){          // закрываем подключение          db.close();          // переход к главной activity          Intent intent = new Intent(this, MainActivity.class);          intent.addFlags(Intent.FLAG\_ACTIVITY\_CLEAR\_TOP | Intent.FLAG\_ACTIVITY\_SINGLE\_TOP);          startActivity(intent);      }  } |

При обновлении или удалении объекта из списка из главной activity в UserActivity будет передаваться id объекта:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | long userId=0;  //.................  Bundle extras = getIntent().getExtras();  if (extras != null) {      userId = extras.getLong("id");  } |

Если из MainActivity не было передано id, то устанавливаем его значение 0, следовательно, у нас будет добавление, а не редактирование/удаление

Если id определен, то получаем по нему из базы данных объект для редактирования/удаления:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | if (id < 0) {        userCursor = db.rawQuery("select \* from " + DatabaseHelper.TABLE + " where " +              DatabaseHelper.COLUMN\_ID + "=?", new String[]{String.valueOf(id)});      userCursor.moveToFirst();      nameBox.setText(userCursor.getString(1));      yearBox.setText(String.valueOf(userCursor.getInt(2)));      userCursor.close();  } |

Иначе просто скрываем кнопку удаления.

Для выполнения операций по вставке, обновлению и удалению данных SQLiteDatabase имеет методы insert(), update() и delete(). Эти методы вызываются в обработчиках кнопок:

|  |  |
| --- | --- |
| 1 | db.delete(DatabaseHelper.TABLE, "\_id = ?", new String[]{String.valueOf(id)}); |

В метод delete() передается название таблицы, а также столбец, по которому происходит удаление, и его значение. В качестве критерия можно выбрать несколько столбцов, поэтому третьим параметром идет массив. Знак вопроса ? обозначает параметр, вместо которого подставляется значение из третьего параметра.

**ContentValues**

Для добавления или обновления нам надо создать объект **ContentValues**. Данный объект представляет словарь, который содержит набор пар "ключ-значение". Для добавления в этот словарь нового объекта применяется метод **put**. Первый параметр метода - это ключ, а второй - значение, например:

|  |  |
| --- | --- |
| 1  2  3 | ContentValues cv = new ContentValues();  cv.put("NAME", "Tom");  cv.put("YEAR", 30); |

В качестве значений в метод put можно передавать строки, целые числа, числа с плавающей точкой

В данном же случае добавляются введенные в текстовое поля значения:

|  |  |
| --- | --- |
| 1  2  3 | ContentValues cv = new ContentValues();  cv.put(DatabaseHelper.COLUMN\_NAME, nameBox.getText().toString());  cv.put(DatabaseHelper.COLUMN\_YEAR, Integer.parseInt(yearBox.getText().toString())); |

При обновлении в метод update() передается название таблицы, объект ContentValues и критерий, по которому происходит обновление (в данном случае столбец id):

|  |  |
| --- | --- |
| 1 | db.update(DatabaseHelper.TABLE, cv, DatabaseHelper.COLUMN\_ID + "=" + String.valueOf(id), null); |

Метод insert() принимает название таблицы, объект ContentValues с добавляемыми значениями. Второй параметр является необязательным: он передает столбец, в который надо добавить значение NULL:

|  |  |
| --- | --- |
| 1 | db.insert(DatabaseHelper.TABLE, null, cv); |

Вместо этих методов, как в прошлой теме, можно использовать метод execSQL() с точным указанием выполняемого sql-выражения. В то же время методы delete/insert/update имеют преимущество - они возвращают id измененной записи, по которому мы можем узнать об успешности операции, или -1 в случае неудачной операции:

|  |  |
| --- | --- |
| 1  2  3  4  5 | long result = db.insert(DatabaseHelper.TABLE, null, cv);  if(result>0){        // действия  } |

После каждой операции выполняется метод goHome(), который возвращает на главную activity.

После этого нам надо исправить код MainActivity, чтобы она инициировала выполнение кода в UserActivity. Для этого изменим код**activity\_main.xml**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <?xml version="1.0" encoding="utf-8"?>  <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="Добавить"          android:onClick="add"          android:textSize="18dp"/>      <ListView          android:id="@+id/list"          android:layout\_width="match\_parent"          android:layout\_height="match\_parent" />  </LinearLayout> |

В данном случае была добавлена кнопка для вызова UserActivity.

И также изменим код класса **MainActivity**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66 | package com.example.eugene.sqliteapp;    import android.content.Intent;  import android.database.Cursor;  import android.database.sqlite.SQLiteDatabase;  import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.view.View;  import android.widget.AdapterView;  import android.widget.ListView;  import android.widget.SimpleCursorAdapter;    public class MainActivity extends AppCompatActivity {        ListView userList;      DatabaseHelper databaseHelper;      SQLiteDatabase db;      Cursor userCursor;      SimpleCursorAdapter userAdapter;        @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          setContentView(R.layout.activity\_main);            userList = (ListView)findViewById(R.id.list);          userList.setOnItemClickListener(new AdapterView.OnItemClickListener() {              @Override              public void onItemClick(AdapterView<?> parent, View view, int position, long id) {                  Intent intent = new Intent(getApplicationContext(), UserActivity.class);                  intent.putExtra("id", id);                  startActivity(intent);              }          });            databaseHelper = new DatabaseHelper(getApplicationContext());      }      @Override      public void onResume() {          super.onResume();          // открываем подключение          db = databaseHelper.getReadableDatabase();            //получаем данные из бд в виде курсора          userCursor =  db.rawQuery("select \* from "+ DatabaseHelper.TABLE, null);          // определяем, какие столбцы из курсора будут выводиться в ListView          String[] headers = new String[] {DatabaseHelper.COLUMN\_NAME, DatabaseHelper.COLUMN\_YEAR};          // создаем адаптер, передаем в него курсор          userAdapter = new SimpleCursorAdapter(this, android.R.layout.two\_line\_list\_item,                  userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2}, 0);          userList.setAdapter(userAdapter);      }      // по нажатию на кнопку запускаем UserActivity для добавления данных      public void add(View view){          Intent intent = new Intent(this, UserActivity.class);          startActivity(intent);      }        @Override      public void onDestroy(){          super.onDestroy();          // Закрываем подключение и курсор          db.close();          userCursor.close();      }  } |

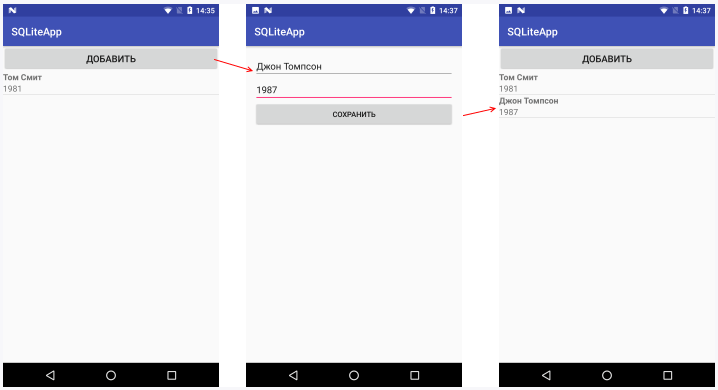
При нажатии на кнопку запускается UserActivity, при этом не передается никакого id, то есть в UserActivity id будет равен нулю, значит будет идти добавление данных:

|  |  |
| --- | --- |
| 1  2  3  4 | public void add(View view){      Intent intent = new Intent(this, UserActivity.class);      startActivity(intent);  } |

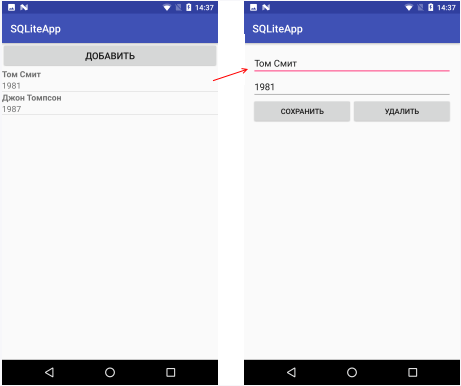
Другую ситуацию представляет обработчик нажатия на элемент списка - при нажатии также будет запускаться UserActivity, но теперь будет передаваться id выбранной записи:

|  |  |
| --- | --- |
| 1  2  3  4  5 | public void onItemClick(AdapterView<?> parent, View view, int position, long id) {      Intent intent = new Intent(getApplicationContext(), UserActivity.class);      intent.putExtra("id", id);      startActivity(intent);  } |

Запустим приложение и нажмем на кнопку, которая должен перенаправлять на UserActivity:



При нажатии в MainActivity на элемент списка этот элемент попадет на UserActivity, где его можно будет удалить или подредактировать:



**Использование существующей БД SQLite**

**Последнее обновление: 05.03.2017**

Кроме создания новой базы данных мы также можем использовать уже существующую. Это может быть более предпочтительно, так как в этом случае база данных приложения уже будет содержать всю необходимую информацию.

Возьмем проект, созданный в предыдущих темах. Для начала создадим базу данных SQLite. В этом нам может помочь такой инструмент как **Sqlitebrowser**. Он бесплатный и доступен для различных операционных систем по адресу <http://sqlitebrowser.org/>. Хотя можно использовать и другие способы для создания начальной БД.

Sqlitebrowser представляет графический интерфейс для создания базы данных и определения в ней всех необходимых таблиц:

Как видно на скриншоте, я определяю таблицу users с тремя полями: \_id, name, age. Общая команда на создание таблицы будет следующей:

|  |  |
| --- | --- |
| 1  2  3  4  5 | CREATE TABLE `users` (      `\_id`   INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,      `name`  TEXT NOT NULL,      `year`  INTEGER NOT NULL  ); |

Там же в программе добавим несколько элементов в созданную таблицу:

После создания таблицы добавим в проект в Android Studio папку **assets**, а в папку assets - только что созданную базу данных. Для этого перейдем к полному опеределению проекта, нажмем на папку main правой кнопкой мыши и в меню выберем **New -> Directory**:

Назовем добавляемую папку assets и затем скопируем в нее нашу базу данных:

В моем случае база данных называется "cityinfo.db".

Изменим код DatabaseHelper следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74 | package com.example.eugene.sqliteapp;    import android.database.SQLException;  import android.database.sqlite.SQLiteOpenHelper;  import android.database.sqlite.SQLiteDatabase;  import android.content.Context;  import android.util.Log;    import java.io.File;  import java.io.FileOutputStream;  import java.io.IOException;  import java.io.InputStream;  import java.io.OutputStream;    class DatabaseHelper extends SQLiteOpenHelper {      private static String DB\_PATH; // полный путь к базе данных      private static String DB\_NAME = "cityinfo.db";      private static final int SCHEMA = 1; // версия базы данных      static final String TABLE = "users"; // название таблицы в бд      // названия столбцов      static final String COLUMN\_ID = "\_id";      static final String COLUMN\_NAME = "name";      static final String COLUMN\_YEAR = "year";      private Context myContext;        DatabaseHelper(Context context) {          super(context, DB\_NAME, null, SCHEMA);          this.myContext=context;          DB\_PATH =context.getFilesDir().getPath() + DB\_NAME;      }        @Override      public void onCreate(SQLiteDatabase db) {      }      @Override      public void onUpgrade(SQLiteDatabase db, int oldVersion,  int newVersion) {      }        void create\_db(){          InputStream myInput = null;          OutputStream myOutput = null;          try {              File file = new File(DB\_PATH);              if (!file.exists()) {                  this.getReadableDatabase();                  //получаем локальную бд как поток                  myInput = myContext.getAssets().open(DB\_NAME);                  // Путь к новой бд                  String outFileName = DB\_PATH;                    // Открываем пустую бд                  myOutput = new FileOutputStream(outFileName);                    // побайтово копируем данные                  byte[] buffer = new byte[1024];                  int length;                  while ((length = myInput.read(buffer)) > 0) {                      myOutput.write(buffer, 0, length);                  }                    myOutput.flush();                  myOutput.close();                  myInput.close();              }          }          catch(IOException ex){              Log.d("DatabaseHelper", ex.getMessage());          }      }      public SQLiteDatabase open()throws SQLException {            return SQLiteDatabase.openDatabase(DB\_PATH, null, SQLiteDatabase.OPEN\_READWRITE);      }  } |

По умолчанию база данных будет размещаться во внешнем хранилище, выделяемом для приложения в папке **/data/data/[название\_пакета]/databases/**, и чтобы получить полный путь к базе данных в конструкторе используется выражение:

|  |  |
| --- | --- |
| 1 | DB\_PATH =context.getFilesDir().getPath() + DB\_NAME; |

Метод onCreate() нам не нужен, так как нам не требуется создание встроенной базы данных. Зато здесь определен дополнительный методcreate\_db(), цель которого копирование базы данных из папки assets в то место, которое указано в переменной DB\_PATH.

Кроме этого здесь также определен метод открытия базы данных open() с помощью метода SQLiteDatabase.openDatabase()

Новый способ организации подключения изменит использование DatabaseHelper в activity. Так, обновим класс **MainActivity**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67 | package com.example.eugene.sqliteapp;    import android.content.Intent;  import android.database.Cursor;  import android.database.sqlite.SQLiteDatabase;  import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.view.View;  import android.widget.AdapterView;  import android.widget.ListView;  import android.widget.SimpleCursorAdapter;    public class MainActivity extends AppCompatActivity {        ListView userList;      DatabaseHelper databaseHelper;      SQLiteDatabase db;      Cursor userCursor;      SimpleCursorAdapter userAdapter;        @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          setContentView(R.layout.activity\_main);            userList = (ListView)findViewById(R.id.list);          userList.setOnItemClickListener(new AdapterView.OnItemClickListener() {              @Override              public void onItemClick(AdapterView<?> parent, View view, int position, long id) {                  Intent intent = new Intent(getApplicationContext(), UserActivity.class);                  intent.putExtra("id", id);                  startActivity(intent);              }          });            databaseHelper = new DatabaseHelper(getApplicationContext());          // создаем базу данных          databaseHelper.create\_db();      }      @Override      public void onResume() {          super.onResume();          // открываем подключение          db = databaseHelper.open();          //получаем данные из бд в виде курсора          userCursor =  db.rawQuery("select \* from "+ DatabaseHelper.TABLE, null);          // определяем, какие столбцы из курсора будут выводиться в ListView          String[] headers = new String[] {DatabaseHelper.COLUMN\_NAME, DatabaseHelper.COLUMN\_YEAR};          // создаем адаптер, передаем в него курсор          userAdapter = new SimpleCursorAdapter(this, android.R.layout.two\_line\_list\_item,                  userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2}, 0);          userList.setAdapter(userAdapter);      }      // по нажатию на кнопку запускаем UserActivity для добавления данных      public void add(View view){          Intent intent = new Intent(this, UserActivity.class);          startActivity(intent);      }        @Override      public void onDestroy(){          super.onDestroy();          // Закрываем подключение и курсор          db.close();          userCursor.close();      }  } |

И также изменим класс **UserActivity**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79 | package com.example.eugene.sqliteapp;    import android.content.ContentValues;  import android.content.Intent;  import android.database.Cursor;  import android.database.sqlite.SQLiteDatabase;  import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.view.View;  import android.widget.Button;  import android.widget.EditText;    public class UserActivity extends AppCompatActivity {        EditText nameBox;      EditText yearBox;      Button delButton;      Button saveButton;        DatabaseHelper sqlHelper;      SQLiteDatabase db;      Cursor userCursor;      long userId=0;      @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          setContentView(R.layout.activity\_user);            nameBox = (EditText) findViewById(R.id.name);          yearBox = (EditText) findViewById(R.id.year);          delButton = (Button) findViewById(R.id.deleteButton);          saveButton = (Button) findViewById(R.id.saveButton);            sqlHelper = new DatabaseHelper(this);          db = sqlHelper.open();            Bundle extras = getIntent().getExtras();          if (extras != null) {              userId = extras.getLong("id");          }          // если 0, то добавление          if (userId > 0) {              // получаем элемент по id из бд              userCursor = db.rawQuery("select \* from " + DatabaseHelper.TABLE + " where " +                      DatabaseHelper.COLUMN\_ID + "=?", new String[]{String.valueOf(userId)});              userCursor.moveToFirst();              nameBox.setText(userCursor.getString(1));              yearBox.setText(String.valueOf(userCursor.getInt(2)));              userCursor.close();          } else {              // скрываем кнопку удаления              delButton.setVisibility(View.GONE);          }      }      public void save(View view){          ContentValues cv = new ContentValues();          cv.put(DatabaseHelper.COLUMN\_NAME, nameBox.getText().toString());          cv.put(DatabaseHelper.COLUMN\_YEAR, Integer.parseInt(yearBox.getText().toString()));            if (userId > 0) {              db.update(DatabaseHelper.TABLE, cv, DatabaseHelper.COLUMN\_ID + "=" + String.valueOf(userId), null);          } else {              db.insert(DatabaseHelper.TABLE, null, cv);          }          goHome();      }      public void delete(View view){          db.delete(DatabaseHelper.TABLE, "\_id = ?", new String[]{String.valueOf(userId)});          goHome();      }      private void goHome(){          // закрываем подключение          db.close();          // переход к главной activity          Intent intent = new Intent(this, MainActivity.class);          intent.addFlags(Intent.FLAG\_ACTIVITY\_CLEAR\_TOP | Intent.FLAG\_ACTIVITY\_SINGLE\_TOP);          startActivity(intent);      }  } |

Вся остальная работа с данными будет той же, чтобы и в прошлых темах:



**Динамический поиск по базе данных SQLite**

**Последнее обновление: 05.03.2017**

Рассмотрим, как мы можем создать в приложении на Android динамический поиск по базе данных SQLite.

Итак, создадим новый проект с пустой MainActivity. Для этого проекта возьмем базу данных из прошлой темы. Данная база данных называется *cityinfo* и имеет одну таблицу users с тремя полями \_id, name, age:

|  |  |
| --- | --- |
| 1  2  3  4  5 | CREATE TABLE `users` (      `\_id`   INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,      `name`  TEXT NOT NULL,      `year`  INTEGER NOT NULL  ); |

И также добавим в проект в Android Studio папку **assets**, а в папку assets - только что созданную базу данных:

В моем случае база данных называется "cityinfo.db".

Как показано выше на скриншоте, добавив в проект в одну папку с MainActivity новый класс DatabaseHelper:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74 | package com.example.eugene.livedbsearchapp;    import android.database.SQLException;  import android.database.sqlite.SQLiteOpenHelper;  import android.database.sqlite.SQLiteDatabase;  import android.content.Context;  import android.util.Log;    import java.io.File;  import java.io.FileOutputStream;  import java.io.IOException;  import java.io.InputStream;  import java.io.OutputStream;    class DatabaseHelper extends SQLiteOpenHelper {      private static String DB\_PATH; // полный путь к базе данных      private static String DB\_NAME = "cityinfo.db";      private static final int SCHEMA = 1; // версия базы данных      static final String TABLE = "users"; // название таблицы в бд      // названия столбцов      static final String COLUMN\_ID = "\_id";      static final String COLUMN\_NAME = "name";      static final String COLUMN\_YEAR = "year";      private Context myContext;        DatabaseHelper(Context context) {          super(context, DB\_NAME, null, SCHEMA);          this.myContext=context;          DB\_PATH =context.getFilesDir().getPath() + DB\_NAME;      }        @Override      public void onCreate(SQLiteDatabase db) {      }      @Override      public void onUpgrade(SQLiteDatabase db, int oldVersion,  int newVersion) {      }        void create\_db(){          InputStream myInput = null;          OutputStream myOutput = null;          try {              File file = new File(DB\_PATH);              if (!file.exists()) {                  this.getReadableDatabase();                  //получаем локальную бд как поток                  myInput = myContext.getAssets().open(DB\_NAME);                  // Путь к новой бд                  String outFileName = DB\_PATH;                    // Открываем пустую бд                  myOutput = new FileOutputStream(outFileName);                    // побайтово копируем данные                  byte[] buffer = new byte[1024];                  int length;                  while ((length = myInput.read(buffer)) > 0) {                      myOutput.write(buffer, 0, length);                  }                    myOutput.flush();                  myOutput.close();                  myInput.close();              }          }          catch(IOException ex){              Log.d("DatabaseHelper", ex.getMessage());          }      }      SQLiteDatabase open()throws SQLException {            return SQLiteDatabase.openDatabase(DB\_PATH, null, SQLiteDatabase.OPEN\_READWRITE);      }  } |

Перейдем к файлу **activity\_main.xml**, который определяет визуальный интерфейс, и изменим его следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent"      android:orientation="vertical">      <EditText android:id="@+id/userFilter"          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content"          android:hint="Поиск" />        <ListView          android:id="@+id/userList"          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content">      </ListView>  </LinearLayout> |

Итак, у нас будет элемент ListView для отображения списка и текстовое поле для фильтрации.

Теперь изменим код **MainActivity**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90 | package com.example.eugene.livedbsearchapp;    import android.database.Cursor;  import android.database.SQLException;  import android.database.sqlite.SQLiteDatabase;  import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.text.Editable;  import android.text.TextWatcher;  import android.widget.EditText;  import android.widget.FilterQueryProvider;  import android.widget.ListView;  import android.widget.SimpleCursorAdapter;    public class MainActivity extends AppCompatActivity {        DatabaseHelper sqlHelper;      SQLiteDatabase db;      Cursor userCursor;      SimpleCursorAdapter userAdapter;      ListView userList;      EditText userFilter;        @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          setContentView(R.layout.activity\_main);          userList = (ListView)findViewById(R.id.userList);          userFilter = (EditText)findViewById(R.id.userFilter);            sqlHelper = new DatabaseHelper(getApplicationContext());          // создаем базу данных          sqlHelper.create\_db();      }      @Override      public void onResume() {          super.onResume();          try {              db = sqlHelper.open();              userCursor = db.rawQuery("select \* from " + DatabaseHelper.TABLE, null);              String[] headers = new String[]{DatabaseHelper.COLUMN\_NAME, DatabaseHelper.COLUMN\_YEAR};              userAdapter = new SimpleCursorAdapter(this, android.R.layout.two\_line\_list\_item,                      userCursor, headers, new int[]{android.R.id.text1, android.R.id.text2}, 0);                // если в текстовом поле есть текст, выполняем фильтрацию              // данная проверка нужна при переходе от одной ориентации экрана к другой              if(!userFilter.getText().toString().isEmpty())                  userAdapter.getFilter().filter(userFilter.getText().toString());                // установка слушателя изменения текста              userFilter.addTextChangedListener(new TextWatcher() {                    public void afterTextChanged(Editable s) { }                    public void beforeTextChanged(CharSequence s, int start, int count, int after) { }                  // при изменении текста выполняем фильтрацию                  public void onTextChanged(CharSequence s, int start, int before, int count) {                        userAdapter.getFilter().filter(s.toString());                  }              });                // устанавливаем провайдер фильтрации              userAdapter.setFilterQueryProvider(new FilterQueryProvider() {                  @Override                  public Cursor runQuery(CharSequence constraint) {                        if (constraint == null || constraint.length() == 0) {                            return db.rawQuery("select \* from " + DatabaseHelper.TABLE, null);                      }                      else {                          return db.rawQuery("select \* from " + DatabaseHelper.TABLE + " where " +                                  DatabaseHelper.COLUMN\_NAME + " like ?", new String[]{"%" + constraint.toString() + "%"});                      }                  }              });                userList.setAdapter(userAdapter);          }          catch (SQLException ex){}      }      @Override      public void onDestroy(){          super.onDestroy();          // Закрываем подключение и курсор          db.close();          userCursor.close();      }  } |

Прежде всего надо отметить, что для фильтрации данных в адаптере, нам надо получить фильтр адаптера, а у этого фильтра выполнить метод filter():

|  |  |
| --- | --- |
| 1 | userAdapter.getFilter().filter(s.toString()); |

В этот метод filter() передается ключ поиска.

Для текстового поля мы можем отслеживать изменения содержимого с помощью слушателя:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | userFilter.addTextChangedListener(new TextWatcher() {        public void afterTextChanged(Editable s) {      }        public void beforeTextChanged(CharSequence s, int start, int count, int after) {      }      // при изменении текста выполняем фильтрацию      public void onTextChanged(CharSequence s, int start, int before, int count) {            userAdapter.getFilter().filter(s.toString());      }  }); |

В слушателе TextWatcher в методе onTextChanged как раз и вызывается метод filter(), в который передется введенная пользователем в текстовое поле последовательность символов.

Сам вызызов метода filter() мало на что влияет. Нам нало еще определить провайдер фильтрации адаптера, которые и будет инкапсулировать реальную логику фильтрации:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | userAdapter.setFilterQueryProvider(new FilterQueryProvider() {      @Override      public Cursor runQuery(CharSequence constraint) {            if (constraint == null || constraint.length() == 0) {                return db.rawQuery("select \* from " + DatabaseHelper.TABLE, null);          }          else {              return db.rawQuery("select \* from " + DatabaseHelper.TABLE + " where " +                  DatabaseHelper.COLUMN\_NAME + " like ?", new String[]{"%" + constraint.toString() + "%"});              }      }  }); |

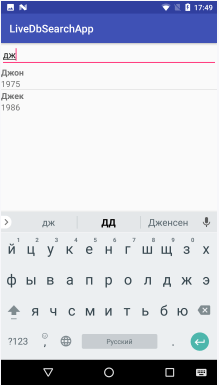
Сущность этого провайдера заключается в выполнении SQL-выражений к бд, а именно конструкций "select from" и "select from where like". Данные простейшие выражения выполняют регистрозависимую фильтрацию. В результате адаптаре получает отфильтрованные данные.

Следует также отметить следующий код:

|  |  |
| --- | --- |
| 1  2 | if(!userFilter.getText().toString().isEmpty())      userAdapter.getFilter().filter(userFilter.getText().toString()); |

Данный код нам нужен при смене ориентации (например, с портретной на альбомную). И если ориентация устройства изменена, но в текстовом поле все же есть некоторые текст-фильтр, то выполняется фильтрация. Иначе бы она не выполнялась.

И после запуска мы сможем насладиться фильтрацией данных:



**Модель, репозиторий и работа с базой данных**

**Последнее обновление: 23.03.2017**

В прошлых темах было рассмотрено взаимодействие с базой данных через класс SimpleCursorAdapter. Но есть и другие способы работы с данными, когда мы абстрагируемся от структуры таблицы и работаем через модель, а все взаимодействие с базой данных производится фактически через реализацию паттерна репозиторий.

Так, создадим новый проект и прежде всего добавим в него класс модели, который назовем **User**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | package com.example.eugene.databaseadapterapp;    public class User {        private long id;      private String name;      private int year;        User(long id, String name, int year){          this.id = id;          this.name = name;          this.year = year;      }      public long getId() {          return id;      }      public String getName() {          return name;      }        public void setName(String name) {          this.name = name;      }        public int getYear() {          return year;      }        public void setYear(int year) {          this.year = year;      }        @Override      public String toString() {          return this.name + " : " + this.year;      }  } |

В данном проекте мы будем работть фактически с теми же данными, что и ранее с данными пользователей, у которых есть уникальный идентификатор, имя и год рождения. И модель User как раз описывает эти данные.

Для взаимодействия с базой данных SQLite добавим новый класс **DatabaseHelper**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | package com.example.eugene.databaseadapterapp;    import android.database.sqlite.SQLiteOpenHelper;  import android.database.sqlite.SQLiteDatabase;  import android.content.Context;  import android.content.ContentValues;    public class DatabaseHelper extends SQLiteOpenHelper {      private static final String DATABASE\_NAME = "userstore.db"; // название бд      private static final int SCHEMA = 1; // версия базы данных      static final String TABLE = "users"; // название таблицы в бд      // названия столбцов      public static final String COLUMN\_ID = "\_id";      public static final String COLUMN\_NAME = "name";      public static final String COLUMN\_YEAR = "year";        public DatabaseHelper(Context context) {          super(context, DATABASE\_NAME, null, SCHEMA);      }        @Override      public void onCreate(SQLiteDatabase db) {            db.execSQL("CREATE TABLE " + TABLE + " (" + COLUMN\_ID                  + " INTEGER PRIMARY KEY AUTOINCREMENT," + COLUMN\_NAME                  + " TEXT, " + COLUMN\_YEAR + " INTEGER);");          // добавление начальных данных          db.execSQL("INSERT INTO "+ TABLE +" (" + COLUMN\_NAME                  + ", " + COLUMN\_YEAR  + ") VALUES ('Том Смит', 1981);");      }      @Override      public void onUpgrade(SQLiteDatabase db, int oldVersion,  int newVersion) {          db.execSQL("DROP TABLE IF EXISTS "+TABLE);          onCreate(db);      }  } |

Также для работы с базой данных добавим в проект класс **DatabaseAdapter**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91 | package com.example.eugene.databaseadapterapp;    import android.content.ContentValues;  import android.content.Context;  import android.database.Cursor;  import android.database.DatabaseUtils;  import android.database.sqlite.SQLiteDatabase;  import java.util.ArrayList;  import java.util.List;    public class DatabaseAdapter {        private DatabaseHelper dbHelper;      private SQLiteDatabase database;        public DatabaseAdapter(Context context){          dbHelper = new DatabaseHelper(context.getApplicationContext());      }        public DatabaseAdapter open(){          database = dbHelper.getWritableDatabase();          return this;      }        public void close(){          dbHelper.close();      }        private Cursor getAllEntries(){          String[] columns = new String[] {DatabaseHelper.COLUMN\_ID, DatabaseHelper.COLUMN\_NAME, DatabaseHelper.COLUMN\_YEAR};          return  database.query(DatabaseHelper.TABLE, columns, null, null, null, null, null);      }        public List<User> getUsers(){          ArrayList<User> users = new ArrayList<>();          Cursor cursor = getAllEntries();          if(cursor.moveToFirst()){              do{                  int id = cursor.getInt(cursor.getColumnIndex(DatabaseHelper.COLUMN\_ID));                  String name = cursor.getString(cursor.getColumnIndex(DatabaseHelper.COLUMN\_NAME));                  int year = cursor.getInt(cursor.getColumnIndex(DatabaseHelper.COLUMN\_YEAR));                  users.add(new User(id, name, year));              }              while (cursor.moveToNext());          }          cursor.close();          return  users;      }        public long getCount(){          return DatabaseUtils.queryNumEntries(database, DatabaseHelper.TABLE);      }        public User getUser(long id){          User user = null;          String query = String.format("SELECT \* FROM %s WHERE %s=?",DatabaseHelper.TABLE, DatabaseHelper.COLUMN\_ID);          Cursor cursor = database.rawQuery(query, new String[]{ String.valueOf(id)});          if(cursor.moveToFirst()){              String name = cursor.getString(cursor.getColumnIndex(DatabaseHelper.COLUMN\_NAME));              int year = cursor.getInt(cursor.getColumnIndex(DatabaseHelper.COLUMN\_YEAR));              user = new User(id, name, year);          }          cursor.close();          return  user;      }        public long insert(User user){            ContentValues cv = new ContentValues();          cv.put(DatabaseHelper.COLUMN\_NAME, user.getName());          cv.put(DatabaseHelper.COLUMN\_YEAR, user.getYear());            return  database.insert(DatabaseHelper.TABLE, null, cv);      }        public long delete(long userId){            String whereClause = "\_id = ?";          String[] whereArgs = new String[]{String.valueOf(userId)};          return database.delete(DatabaseHelper.TABLE, whereClause, whereArgs);      }        public long update(User user){            String whereClause = DatabaseHelper.COLUMN\_ID + "=" + String.valueOf(user.getId());          ContentValues cv = new ContentValues();          cv.put(DatabaseHelper.COLUMN\_NAME, user.getName());          cv.put(DatabaseHelper.COLUMN\_YEAR, user.getYear());          return database.update(DatabaseHelper.TABLE, cv, whereClause, null);      }  } |

Фактически данный класс выполняет роль репозитория данных. Чтобы взамодействовать с БД он определяет методы open() и close(), которые соответственно открывают и закрывают подключение к базе данных.

Непосредственно для работы с данными в классе определены методы insert() (добавление), delete() (удаление), update() (обновление), getUsers() (получение всех пользователей из таблицы) и getUser() (получение одного пользователя по id).

В качестве пользовательского интерфейса будем отталкиваться от того функционала, который использовался в прошлых темах. Так, добавим в проект новый класс Activity - **UserActivity**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77 | package com.example.eugene.databaseadapterapp;    import android.content.Intent;  import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.view.View;  import android.widget.Button;  import android.widget.EditText;    public class UserActivity extends AppCompatActivity {        private EditText nameBox;      private EditText yearBox;      private Button delButton;      private Button saveButton;        private DatabaseAdapter adapter;      private long userId=0;      @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          setContentView(R.layout.activity\_user);            nameBox = (EditText) findViewById(R.id.name);          yearBox = (EditText) findViewById(R.id.year);          delButton = (Button) findViewById(R.id.deleteButton);          saveButton = (Button) findViewById(R.id.saveButton);          adapter = new DatabaseAdapter(this);            Bundle extras = getIntent().getExtras();          if (extras != null) {              userId = extras.getLong("id");          }          // если 0, то добавление          if (userId > 0) {              // получаем элемент по id из бд              adapter.open();              User user = adapter.getUser(userId);              nameBox.setText(user.getName());              yearBox.setText(String.valueOf(user.getYear()));              adapter.close();          } else {              // скрываем кнопку удаления              delButton.setVisibility(View.GONE);          }      }        public void save(View view){              String name = nameBox.getText().toString();          int year = Integer.parseInt(yearBox.getText().toString());          User user = new User(userId, name, year);            adapter.open();          if (userId > 0) {              adapter.update(user);          } else {              adapter.insert(user);          }          adapter.close();          goHome();      }      public void delete(View view){            adapter.open();          adapter.delete(userId);          adapter.close();          goHome();      }      private void goHome(){          // переход к главной activity          Intent intent = new Intent(this, MainActivity.class);          intent.addFlags(Intent.FLAG\_ACTIVITY\_CLEAR\_TOP | Intent.FLAG\_ACTIVITY\_SINGLE\_TOP);          startActivity(intent);      }  } |

Эта activity используется для добавления/редактирования/удаления одного объекта User. Если в UserActivity передается параметр id, то значит мы находимся в режиме редактирования пользователя, поэтому обращаемся к методу getUser() класса DatabaseAdapter для получения нужного пользователя.

Для добавления/изменения/удаления пользователя по нажатию на кнопку вызывается соответствующий метод класса DatabaseAdapter.

В файле **activity\_user.xml** в папке *res/layout* определим для UserActivity простейший интерфейс:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | <?xml version="1.0" encoding="utf-8"?>  <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"      android:orientation="vertical"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent"      android:padding="16dp">      <EditText          android:id="@+id/name"          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content"          android:hint="Введите имя"/>      <EditText          android:id="@+id/year"          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content"          android:hint="Введите год рождения"/>      <LinearLayout          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content"          android:orientation="horizontal">            <Button              android:id="@+id/saveButton"              android:layout\_width="0dp"              android:layout\_weight="1"              android:layout\_height="wrap\_content"              android:text="Сохранить"              android:onClick="save"/>          <Button              android:id="@+id/deleteButton"              android:layout\_width="0dp"              android:layout\_weight="1"              android:layout\_height="wrap\_content"              android:text="Удалить"              android:onClick="delete"/>      </LinearLayout>  </LinearLayout> |

В файле **activity\_main.xml** в папке *res/layout* определим визуальный интерфейс для MainActivity:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <?xml version="1.0" encoding="utf-8"?>  <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="Добавить"          android:onClick="add"          android:textSize="18sp"/>      <ListView          android:id="@+id/list"          android:layout\_width="match\_parent"          android:layout\_height="match\_parent" />  </LinearLayout> |

Здесь имеется элемент ListView для вывода объектов из таблицы и кнопка для перехода к UserActivity для добавления пользователя.

И изменим код **MainActivity**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | package com.example.eugene.databaseadapterapp;    import android.content.Intent;  import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.view.View;  import android.widget.AdapterView;  import android.widget.ArrayAdapter;  import android.widget.ListView;    import java.util.List;    public class MainActivity extends AppCompatActivity {        private ListView userList;      ArrayAdapter<User> arrayAdapter;      @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          setContentView(R.layout.activity\_main);            userList = (ListView)findViewById(R.id.list);            userList.setOnItemClickListener(new AdapterView.OnItemClickListener() {              @Override              public void onItemClick(AdapterView<?> parent, View view, int position, long id) {                  User user =arrayAdapter.getItem(position);                  if(user!=null) {                      Intent intent = new Intent(getApplicationContext(), UserActivity.class);                      intent.putExtra("id", user.getId());                      intent.putExtra("click", 25);                      startActivity(intent);                  }              }          });      }        @Override      public void onResume() {          super.onResume();          DatabaseAdapter adapter = new DatabaseAdapter(this);          adapter.open();            List<User> users = adapter.getUsers();            arrayAdapter = new ArrayAdapter<>(this, android.R.layout.simple\_list\_item\_1, users);          userList.setAdapter(arrayAdapter);          adapter.close();      }      // по нажатию на кнопку запускаем UserActivity для добавления данных      public void add(View view){          Intent intent = new Intent(this, UserActivity.class);          startActivity(intent);      }  } |

В переопределенном методе onResume() через объект DatabaseAdapter получаем всех пользователей из базы данных и через ArrayAdapter выводим их в ListView.

При нажатии на элемент ListView запускаем UserActivity, передавая ей id выделенного пользователя.

При нажатии на кнопку просто вызываем UserActivity.

В итоге весь проект будет выглядеть следующим образом:

При запуске MainActivity отобразит список пользователей из базы данных, а при переходе к UserActivity мы сможем подредактировать или добавить пользователей:

