PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS

Uso do SQLite no Android

Professor: Danilo Giacobo

OBJETIVOS DA AULA

- Aprender a persistir dados utilizando o banco de dados SQLite.
- Conhecer e utilizar a classe SQLiteOpenHelper.

INTRODUÇÃO

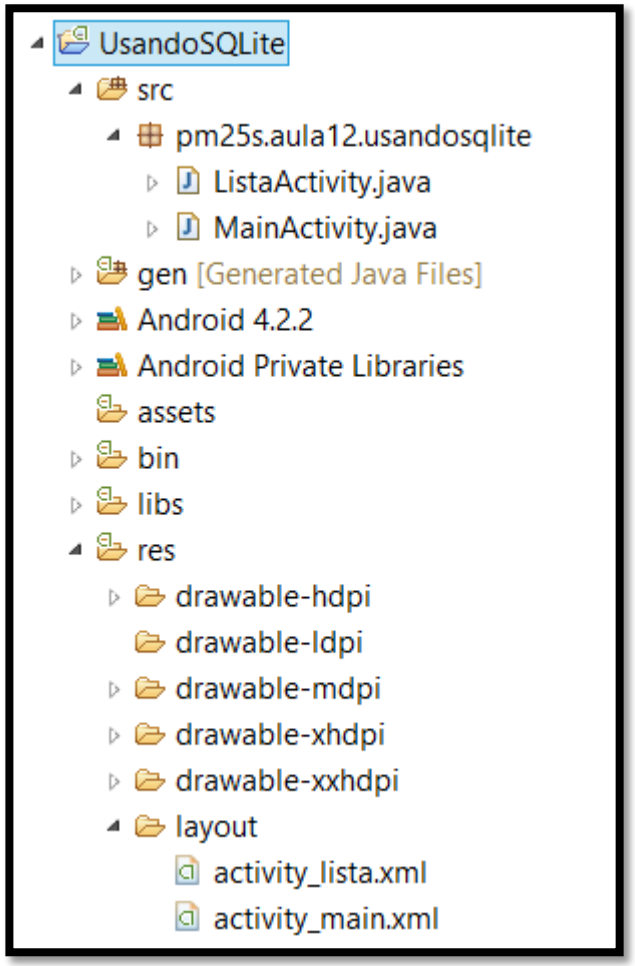
- O uso dos *smartphones* cresceu muito nos últimos anos.
- Os usuários passaram a armazenar seus dados não só nos computadores *desktop*, mas também em seus *smartphones*.
- Atualmente, as principais plataformas oferecem mais de uma maneira de salvar e recuperar as informações produzidas por nossos aplicativos.
- A plataforma Android oferece diferentes métodos de persistência:
  - Armazenamento interno (memória do smartphone)
  - Armazenamento externo (cartão de memória)
  - Classe *SharedPreferences*
  - Bancos de Dados Relacionais (SQLite)

SQLITE 

- O SQLite é um banco de dados simples e poderoso, formatado para ser executado na plataforma móvel.
- É um repositório que simula um banco de dados relacional.
- Os dados são armazenados de forma binária (transparente para o usuário).
- É a principal forma de persistência nos aplicativos Android.
- O SQLite é formado por um conjunto de bibliotecas escritas em C.
- É um banco de dados gratuito.
- O Android oferece suporte completo ao banco por meio de uma API com um rico conjunto de classes e métodos que abstraem as complexidades dos códigos SQL.

O PROJETO

- Para exemplificar o uso do SQLite, um aplicativo de cadastro de notas relativamente simples será desenvolvido.
- Crie um projeto chamado UsandoSQLite.
- Além da *Activity* principal, crie uma nova *Activity* para apresentar as informações do banco em uma lista, chamada de **ListaActivity.java**, com uma tela **activity_lista.xml**, conforme imagem ao lado.



O PROJETO

- A tela principal do aplicativo será formada por três campos: **Código**, **Nome da Disciplina** e **Nota**, além de cinco botões: **Incluir**, **Alterar**, **Excluir**, **Pesquisar** e **Listar**.
- O campo da chave primária do nosso aplicativo é o campo código, sendo este obrigatório para as operações de alterar, excluir e pesquisar.
- Para a operação de incluir, esse campo não é necessário, uma vez que será AUTOINCREMENT na tabela.
- A opção de listar apresenta todos os dados do banco. Para esta operação o campo código não precisa ser preenchido.
- O aplicativo, embora simples, procura apresentar as funcionalidades do banco de dados SQLite, sendo executado a partir de métodos específicos para a inclusão, alteração, e assim por diante.
- Existe uma maneira mais elaborada de usar o SQLite a partir da classe *SQLiteOpenHelper*.

O PROJETO - INTERFACE GRÁFICA PRINCIPAL

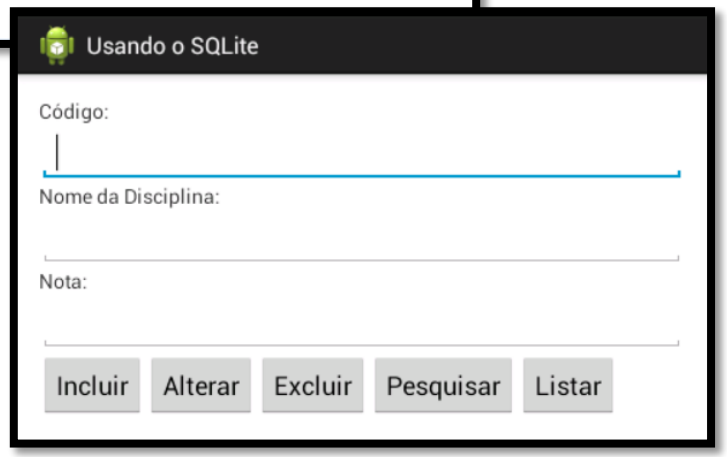
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/codigo" />

    <EditText
        android:id="@+id/etCodigo"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:inputType="number" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/nome_disciplina" />

    <EditText
        android:id="@+id/etNomeDisciplina"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:inputType="textCapWords" />
```

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/nota" />

    <EditText
        android:id="@+id/etNota"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:inputType="numberDecimal" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <Button
           android:id="@+id/btIncluir"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/incluir"
            android:onClick="btIncluirOnClick" />

        <Button
           android:id="@+id/btAlterar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/alterar"
            android:onClick="btAlterarOnClick" />
```

O PROJETO - INTERFACE GRÁFICA PRINCIPAL

```
64          <Button
65              android:id="@+id/btExcluir"
66               android:layout_width="wrap_content"
67               android:layout_height="wrap_content"
68               android:text="@string/excluir"
69               android:onClick="btExcluirOnClick" />
70
71          <Button
72              android:id="@+id/btPesquisar"
73               android:layout_width="wrap_content"
74               android:layout_height="wrap_content"
75               android:text="@string/pesquisar"
76               android:onClick="btPesquisarOnClick" />
77
78          <Button
79              android:id="@+id/btListar"
80               android:layout_width="wrap_content"
81               android:layout_height="wrap_content"
82               android:text="@string/listar"
83               android:onClick="btListarOnClick" />
84
85      </LinearLayout>
86
87  </LinearLayout>
```

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3
4      <string name="app_name">Usando o SQLite</string>
5      <string name="action_settings">Settings</string>
6      <string name="codigo">Código: </string>
7      <string name="nome_disciplina">Nome da Disciplina: </string>
8      <string name="nota">Nota: </string>
9      <string name="incluir">Incluir</string>
10     <string name="alterar">Alterar</string>
11     <string name="excluir">Excluir</string>
12     <string name="pesquisar">Pesquisar</string>
13     <string name="listar">Listar</string>
14
15     <string name="title_activity_lista">ListaActivity</string>
16
17 </resources>
```



O PROJETO - CLASSE PRINCIPAL

```java
package pm25s.aula12.usandosqlite;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

public class MainActivity extends Activity {

    private EditText etCodigo;
    private EditText etNomeDisciplina;
    private EditText etNota;

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

        etCodigo = (EditText) findViewById(R.id.etCodigo);
        etNomeDisciplina = (EditText) findViewById(R.id.etNomeDisciplina);
        etNota = (EditText) findViewById(R.id.etNota);
    }
```

```java
    public void btIncluirOnClick(View v) {

    }

    public void btAlterarOnClick(View v) {

    }

    public void btExcluirOnClick(View v) {

    }

    public void btPesquisarOnClick(View v) {

    }

    public void btListarOnClick(View v) {

    }
}
```


O PROJETO - INTERFACE GRÁFICA DE LISTAR

- A interface gráfica **activity_lista.xml** é composta exclusivamente de um componente ListView, o qual apresentará os elementos do banco de dados dispostos em uma lista, conforme apresentado abaixo:

```
<ListView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:id="@+id/lvListar"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".ListaActivity" >

</ListView>
```

O PROJETO - CLASSE DE LISTAR

- O código da *Activity* que trata a interface anterior recupera o *ListView* para a formatação do seu conteúdo, conforme apresentado abaixo:

```
1  package pm25s.aula12.usandosqlite;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5  import android.widget.ListView;
6
7  public class ListaActivity extends Activity {
8
9      private ListView lvRegistros;
10
11     @Override
12     protected void onCreate(Bundle savedInstanceState) {
13         super.onCreate(savedInstanceState);
14         setContentView(R.layout.activity_lista);
15
16         lvRegistros = (ListView) findViewById(R.id.lvListar);
17     }
18 }
```

USANDO O SQLITE NO ANDROID

- Para fazer uso do SQLite no Android, o primeiro passo é a instanciação de um objeto do tipo SQLiteDatabase, sendo este o objeto que fará a comunicação com o banco de dados, permitindo a execução de comandos, como, por exemplo, a criação de tabelas, manipulação e consulta de registros. Este código é apresentado abaixo:

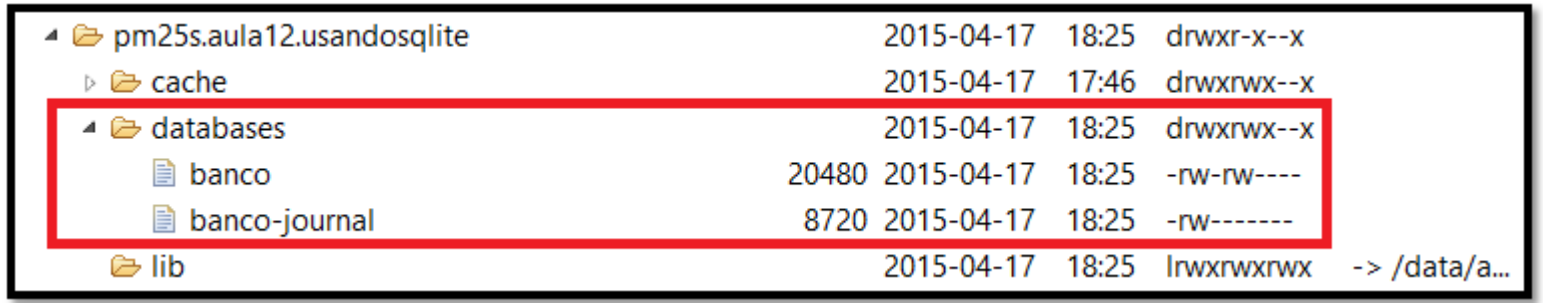
```
import android.database.sqlite.SQLiteDatabase;
```

```
private SQLiteDatabase banco;
```

```
banco = this.openOrCreateDatabase("banco", Context.MODE_PRIVATE, null);
banco.execSQL("CREATE TABLE IF NOT EXISTS notas (_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
              "nome_disciplina TEXT NOT NULL, nota DECIMAL NOT NULL)");
```

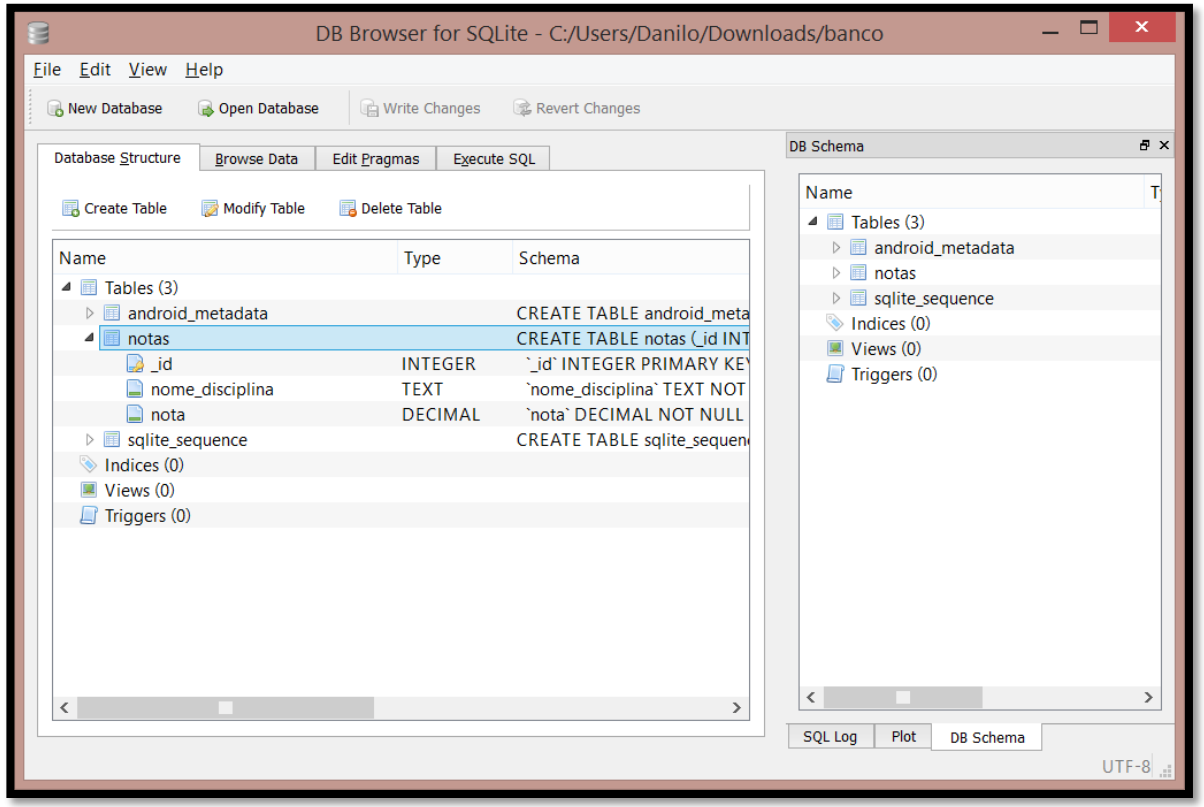
MODO DE UTILIZAÇÃO DO SQLITE

- Não é comum a manipulação do banco de dados do Android como acontece na programação *desktop* e *Web*, inclusive ferramentas como o MySQL-Front, PGAdmin e IBExpert não são comuns para o Android, já que o usuário não costuma manipular diretamente os dados. A maneira mais comum de manipular os dados é via linha de código.
- Desta forma, a criação do banco e da tabela costuma ser feita no método construtor da tela que usa pela primeira vez essa base de dados, entretanto, após a criação do banco, o mesmo é armazenado na **pasta data/data** do *device*, dentro do pacote do projeto, como pode ser visto abaixo:



MODO DE UTILIZAÇÃO DO SQLITE

- Se o usuário fizer questão, o mesmo poderá recuperar o arquivo do banco de dados, apresentado em destaque na figura anterior, e utilizar alguns programas desktop para manipular o banco SQLite, como, por exemplo, o SQLite Data Browser, uma ferramenta gratuita e multiplataforma cuja tela é apresentada ao lado.



O PROJETO - MANIPULAÇÃO DO BANCO DE DADOS

- O passo seguinte é fazer os comandos de manipulação do banco de dados na interface gráfica principal. A listagem abaixo apresenta o código referente ao comando de *Inclusão*.

```
import android.content.ContentValues;
```

```
public void btIncluirOnClick(View v) {
    ContentValues registro = new ContentValues();
    registro.put("nome_disciplina", etNomeDisciplina.getText().toString());
    registro.put("nota", Double.parseDouble(etNota.getText().toString()));
    banco.insert("notas", null, registro);
    Toast.makeText(getApplicationContext(), "Sucesso!", Toast.LENGTH_SHORT).show();
}
```

- O método **insert()** retorna o **ID da linha** criada no banco de dados ou **-1** se um erro ocorreu.

O PROJETO - MANIPULAÇÃO DO BANCO DE DADOS

- A listagem abaixo apresenta o código referente ao comando de *Alteração*.

```
public void btAlterarOnClick(View v) {
    int id = Integer.parseInt(etCodigo.getText().toString());
    ContentValues registro = new ContentValues();
    registro.put("nome_disciplina", etNomeDisciplina.getText().toString());
    registro.put("nota", Double.parseDouble(etNota.getText().toString()));
    banco.update("notas", registro, "_id = " + id, null);
    Toast.makeText(getApplicationContext(), "Sucesso!", Toast.LENGTH_SHORT).show();
}
```

- Outra forma de usar o método de alteração:

```
banco.update("notas", registro, "_id = ?", new String[] {String.valueOf(id)});
```

- O método **update()** retorna o número de linhas afetadas.

O PROJETO - MANIPULAÇÃO DO BANCO DE DADOS

- A listagem abaixo apresenta o código referente ao comando de *Exclusão*.

```
public void btExcluirOnClick(View v) {
    int id = Integer.parseInt(etCodigo.getText().toString());
    banco.delete("notas", "_id = " + id, null);
    Toast.makeText(getApplicationContext(), "Sucesso!", Toast.LENGTH_SHORT).show();
}
```

Dica: Utilizando a sintaxe SQL

Aos programadores adeptos da linguagem SQL, é possível utilizá-la na inclusão e para isso, basta utilizar o método **execSQL()** no objeto do tipo SQLiteDatabase. Para a inclusão, a sintaxe seria a seguinte:

```
banco.execSQL("INSERT INTO notas (nome_disciplina, nota) VALUES ('" +
              etNomeDisciplina.getText().toString() + "', " + etNota.getText().toString() + ")");
```

O PROJETO - MANIPULAÇÃO DO BANCO DE DADOS

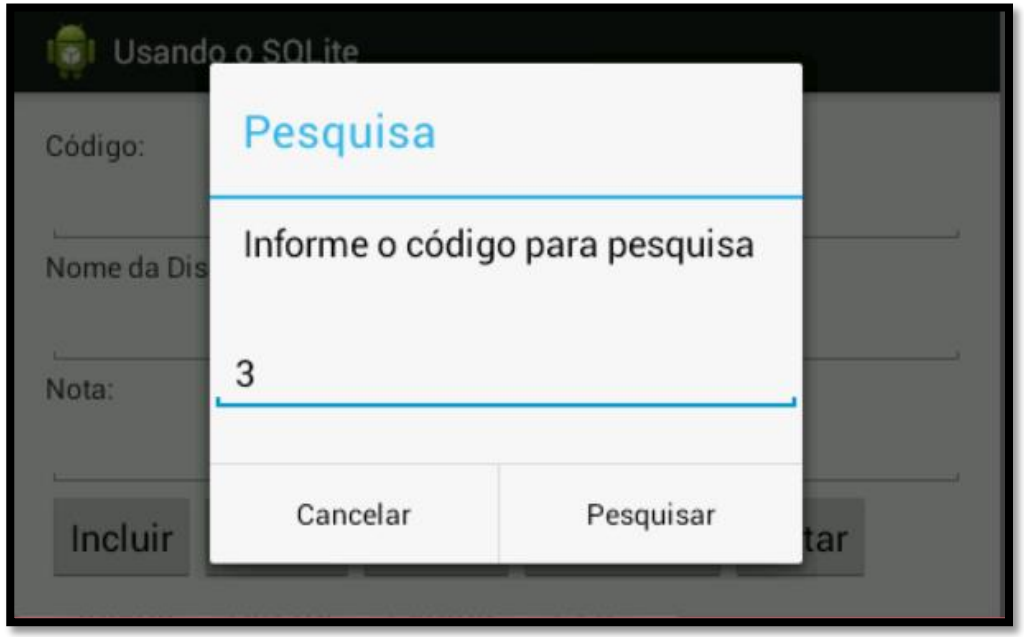
- O método **delete()** retorna o número de linhas afetadas se o segundo parâmetro for informado (cláusula WHERE).
- A próxima funcionalidade mostrada no sistema é a opção de pesquisa que, ao ser escolhida, apresentará para o usuário uma janela interna (*dialog*), solicitando o código do registro a ser pesquisado. Ao confirmar, os dados recuperados do banco de dados serão apresentados nos campos da tela.
- O código dessa funcionalidade é exibido no próximo slide.

O PROJETO - MANIPULAÇÃO DO BANCO DE DADOS

```
public void btPesquisarOnClick(View v) {
    final EditText etCodPesquisa = new EditText(getApplicationContext());
    etCodPesquisa.setTextColor(Color.BLACK);

    AlertDialog.Builder telaPesquisa = new AlertDialog.Builder(this);
    telaPesquisa.setTitle("Pesquisa");
    telaPesquisa.setMessage("Informe o código para pesquisa");
    telaPesquisa.setView(etCodPesquisa);
    telaPesquisa.setNegativeButton("Cancelar", null);
    telaPesquisa.setPositiveButton("Pesquisar", new DialogInterface.OnClickListener() {

        @Override
        public void onClick(DialogInterface dialog, int which) {
            realizarPesquisa(Integer.parseInt(etCodPesquisa.getText().toString()));
        }
    });

    telaPesquisa.show();
}
```

```
protected void realizarPesquisa(int id) {
    Cursor registros = banco.query("notas", null, "_id = " + id, null, null, null, null);

    if(registros.moveToNext()) {
        String nomeDisciplina = registros.getString(registros.getColumnIndex("nome_disciplina"));
        double nota = registros.getDouble(registros.getColumnIndex("nota"));

        etCodigo.setText(String.valueOf(id));
        etNomeDisciplina.setText(nomeDisciplina);
        etNota.setText(String.valueOf(nota));

        Toast.makeText(getApplicationContext(), "Sucesso!", Toast.LENGTH_SHORT).show();
    } else {
        Toast.makeText(getApplicationContext(), "Registro não encontado!", Toast.LENGTH_SHORT).show();
    }
}
```

O PROJETO - PESQUISA EM AÇÃO



O PROJETO - MANIPULAÇÃO DO BANCO DE DADOS

Dica: Utilizando a sintaxe SQL

- Assim como os comandos de manipulação de dados (insert, update e delete), os comandos de pesquisa também podem ser executados a partir da sintaxe SQL. Um exemplo é apresentado no código abaixo:

```
Cursor registros = banco.rawQuery("SELECT * FROM notas WHERE _id = " + id, null);
```

- O primeiro parâmetro do método **rawQuery** é a sintaxe SQL do comando SQL, o segundo parâmetro é um array de strings com o conteúdo que substituirá o parâmetro ? Presente no comando SQL.

O PROJETO - MANIPULAÇÃO DO BANCO DE DADOS

- O último passo no desenvolvimento da nossa aplicação é tratar a listagem dos dados, que apresentará uma nova tela com todos os registros existentes e, desta forma, a lógica codificada na classe principal para listar se resume à chamada de uma nova tela conforme apresentado abaixo:

```
import android.content.Intent;
```

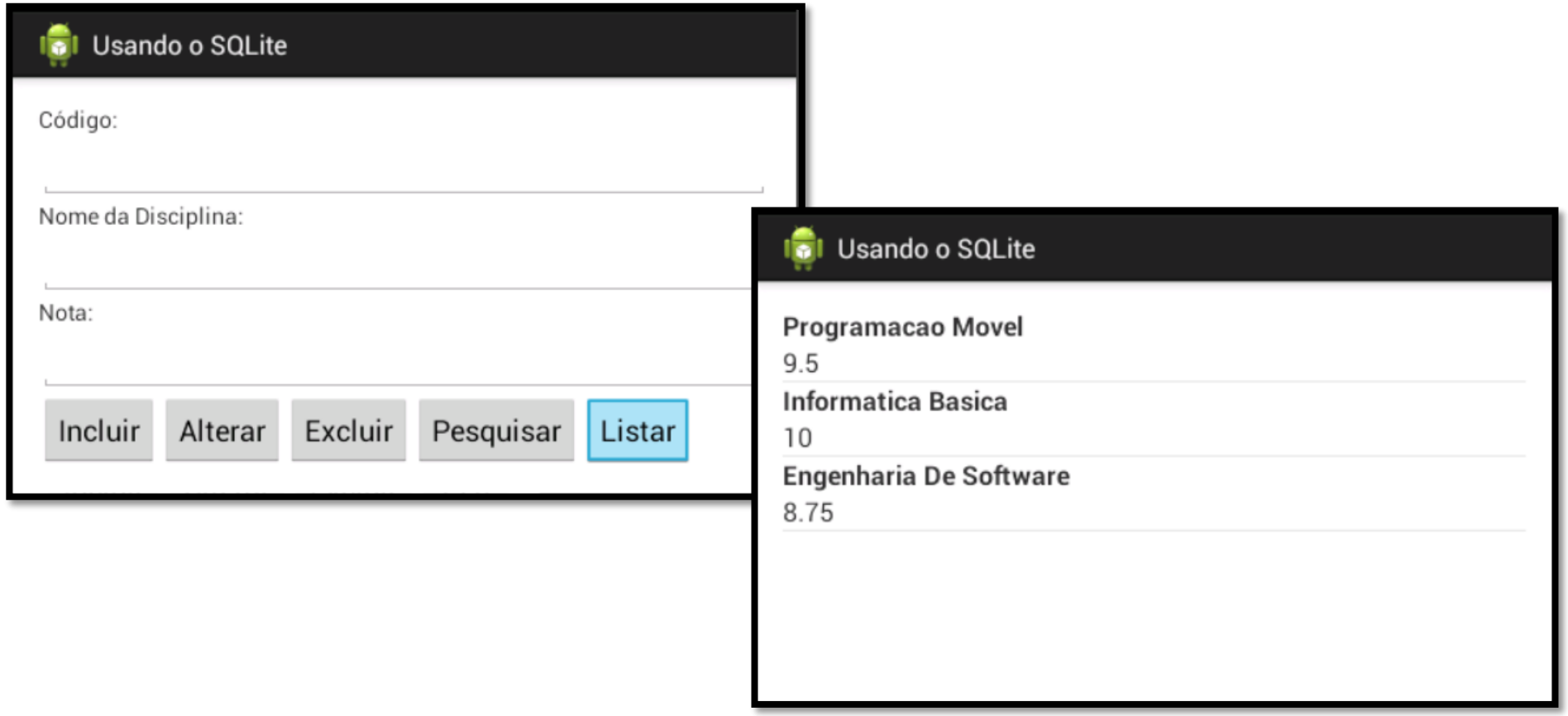
```
public void btListarOnClick(View v) {
    startActivity(new Intent(getApplicationContext(), ListaActivity.class));
}
```

- Para concluir a codificação a classe que lista os dados, devemos acessar o SQLite, recuperar todos os registros e adicioná-los ao componente *ListView*. Existe um **Adapter** chamado **SimpleCursorAdapter** que facilita este processo. O código completo da classe de listagem é mostrado no próximo slide.

O PROJETO - MANIPULAÇÃO DO BANCO DE DADOS

```
1  package pm25s.aula12.usandosqlite;
2
3  import android.app.Activity;
10
11 public class ListaActivity extends Activity {
12
13     private ListView lvRegistros;
14     private SQLiteDatabase bd;
15
16     @Override
17     protected void onCreate(Bundle savedInstanceState) {
18         super.onCreate(savedInstanceState);
19         setContentView(R.layout.activity_lista);
20
21         lvRegistros = (ListView) findViewById(R.id.lvListar);
22         bd = this.openOrCreateDatabase("banco", Context.MODE_PRIVATE, null);
23         montarListaRegistro();
24     }
25
26     @SuppressWarnings("deprecation")
27     private void montarListaRegistro() {
28         Cursor registros = bd.query("notas", null, null, null, null, null, null);
29         String nomeCamposTabela[] = new String[] {"nome_disciplina", "nota"};
30         int nomeCamposTela[] = new int[] {android.R.id.text1, android.R.id.text2};
31         SimpleCursorAdapter adapter = new SimpleCursorAdapter(getBaseContext(),
32                                                               android.R.layout.two_line_list_item,
33                                                               registros,
34                                                               nomeCamposTabela,
35                                                               nomeCamposTela);
36         lvRegistros.setAdapter(adapter);
37     }
```

O PROJETO - MANIPULAÇÃO DO BANCO DE DADOS



O PROJETO - MANIPULAÇÃO DO BANCO DE DADOS

Dica: Apresentando um elemento por linha

- Se o objetivo é apresentar apenas um elemento por linha, como, por exemplo, o campo *nome_disciplina*, a instanciação do *SimpleCursorAdapter* pode ser alterada, conforme o exemplo abaixo:

```
private void montarListaRegistro() {
    Cursor registros = bd.query("notas", null, null, null, null, null, null);
    String nomeCamposTabela[] = new String[] {"nome_disciplina"};
    int nomeCamposTela[] = new int[] {android.R.id.text1};
    SimpleCursorAdapter adapter = new SimpleCursorAdapter(getBaseContext(),
                                                          android.R.layout.simple_list_item_1,
                                                          registros,
                                                          nomeCamposTabela,
                                                          nomeCamposTela);

    lvRegistros.setAdapter(adapter);
}
```

SQLITE OPEN HELPER

- Para uma melhor organização do código, muitos programadores Android utilizam um modelo baseado na classe **SQLiteOpenHelper,** sendo a partir desta criada uma subclasse que possui as funcionalidades do banco de dados.
- O objetivo principal dessa classe é criar uma classe específica para a manipulação do banco de dados. Esta classe terá uma instância do **SQLiteDatabase.**
- Iremos a partir de agora alterar nosso projeto desenvolvido até então para usar a classe **SQLiteOpenHelper**.
- O primeiro passo para o uso desta classe é a criação de uma **classe de entidade,** contendo uma imagem da tabela utilizada. É aconselhável criar um *pacote* para armazenar as classes de entidades.
- O código da classe de entidade **Notas** é apresentado no próximo slide.

SQLITE OPEN HELPER

```java
package pm25s.aula12.entidades;

public class Notas {

    private int _id;
    private String nomeDisciplina;
    private double nota;

    public Notas() {

    }

    public Notas(int _id, String nomeDisciplina, double nota) {
        this._id = _id;
        this.nomeDisciplina = nomeDisciplina;
        this.nota = nota;
    }

    public Notas(String nomeDisciplina, double nota) {
        this.nomeDisciplina = nomeDisciplina;
        this.nota = nota;
    }
```

```java
    public int get_id() {
        return _id;
    }

    public void set_id(int _id) {
        this._id = _id;
    }

    public String getNomeDisciplina() {
        return nomeDisciplina;
    }

    public void setNomeDisciplina(String nomeDisciplina) {
        this.nomeDisciplina = nomeDisciplina;
    }

    public double getNota() {
        return nota;
    }

    public void setNota(double nota) {
        this.nota = nota;
    }
}
```

SQLITE OPEN HELPER

Dica: Classes de entidade

- Ao utilizar a classe *SQLiteOpenHelper*, aconselha-se a codificação de uma classe de entidade para cada tabela no banco de dados. Em nosso projeto, apenas um classe foi codificada, pois existe apenas uma tabela no banco. Se existissem duas tabelas, duas classes de entidades deveriam ser codificadas, cada uma representando uma tabela. As classes de entidade farão a troca de informações entre a classe *Activity* e a classe-filha de *SQLiteOpenHelper*.
- O próximo passo em nosso projeto atual é criar uma classe para a manipulação do banco de dados, que costuma ficar em outro pacote. O código desta classe é mostrado no próximo slide.

SQLITE OPEN HELPER

```
1  package pm25s.aula12.bd;
2
3  import android.content.Context;
4  import android.database.sqlite.SQLiteDatabase;
5  import android.database.sqlite.SQLiteOpenHelper;
6
7  public class DatabaseHandler extends SQLiteOpenHelper {
8
9      private static final int DATABASE_VERSION = 1;
10     private static final String DATABASE_NAME = "banco";
11
12     public DatabaseHandler(Context context) {
13         super(context, DATABASE_NAME, null, DATABASE_VERSION);
14     }
15
16     @Override
17     public void onCreate(SQLiteDatabase banco) {
18         banco.execSQL("CREATE TABLE notas1 (_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
19                       "nome_disciplina TEXT NOT NULL, nota DECIMAL NOT NULL)");
20     }
21
22     @Override
23     public void onUpgrade(SQLiteDatabase banco, int oldVersion, int newVersion) {
24         banco.execSQL("DROP TABLE IF EXISTS notas1");
25         onCreate(banco);
26     }
27 }
```

SQLITE OPEN HELPER

- A estrutura apresentada no slide anterior é a estrutura mínima de uma subclasse de SQLiteOpenHelper. O próximo passo é codificar os métodos para a manutenção do banco de dados. O primeiro deles é o Incluir, mostrado abaixo:

```
public void incluirRegistro(Notas notas) {
    SQLiteDatabase banco = this.getWritableDatabase();

    ContentValues registro = new ContentValues();
    registro.put("nome_disciplina", notas.getNomeDisciplina());
    registro.put("nota", notas.getNota());

    banco.insert("notas1", null, registro);
    banco.close();
}
```

SQLITE OPEN HELPER

- O código dos métodos Alterar e Excluir é apresentado abaixo:

```
public void alterarRegistro(Notas notas) {
    SQLiteDatabase banco = this.getWritableDatabase();

    ContentValues registro = new ContentValues();
    registro.put("nome_disciplina", notas.getNomeDisciplina());
    registro.put("nota", notas.getNota());

    banco.update("notas1", registro, "_id = " + notas.get_id(), null);
    banco.close();
}
```

```
public void excluirRegistro(int id) {
    SQLiteDatabase banco = this.getWritableDatabase();
    banco.delete("notas1", "_id = ?", new String[] {String.valueOf(id)});
    banco.close();
}
```

SQLITE OPEN HELPER

- O código dos métodos Pesquisar e Listar é apresentado abaixo:

```
public Notas pesquisarRegistro(int id) {
    SQLiteDatabase banco = this.getWritableDatabase();

    Cursor registros = banco.query("notas1", null, "_id = " + id, null, null, null, null);

    if(registros.moveToNext()) {
        String nomeDisciplina = registros.getString(registros.getColumnIndex("nome_disciplina"));
        double nota = registros.getDouble(registros.getColumnIndex("nota"));

        Notas notas = new Notas();
        notas.set_id(id);
        notas.setNomeDisciplina(nomeDisciplina);
        notas.setNota(nota);

        return notas;
    } else {
        return null;
    }
}
```

```
public Cursor listarRegistros() {
    SQLiteDatabase banco = this.getWritableDatabase();

    Cursor registros = banco.query("notas1", null, null, null, null, null, null);

    return registros;
}
```

SQLITE OPEN HELPER

- Será necessário alterar a classe principal da aplicação para utilizar a nova classe de manipulação de banco de dados criada anteriormente. O novo código da classe é mostrado por partes a partir desse slide e dos slides seguintes:

```
1  package pm25s.aula12.usandosqlite;
2
3  import android.app.Activity;
4  import android.app.AlertDialog;
5  import android.content.ContentValues;
6  import android.content.DialogInterface;
7  import android.content.Intent;
8  import android.database.Cursor;
9  import android.os.Bundle;
10 import android.view.View;
11 import android.widget.EditText;
12 import android.widget.Toast;
13 import pm25s.aula12.bd.DatabaseHandler;
14 import pm25s.aula12.entidades.Notas;
15
16 public class MainActivity extends Activity {
17
18     private EditText etCodigo;
19     private EditText etNomeDisciplina;
20     private EditText etNota;
21
22     private DatabaseHandler banco;
```

```
24     @Override
25     protected void onCreate(Bundle savedInstanceState) {
26         super.onCreate(savedInstanceState);
27         setContentView(R.layout.activity_main);
28
29         etCodigo = (EditText) findViewById(R.id.etCodigo);
30         etNomeDisciplina = (EditText) findViewById(R.id.etNomeDisciplina);
31         etNota = (EditText) findViewById(R.id.etNota);
32
33         banco = new DatabaseHandler(this);
34     }
```

SQLITE OPEN HELPER

```
36    public void btIncluirOnClick(View v) {
37        Notas notas = new Notas();
38        notas.setNomeDisciplina(etNomeDisciplina.getText().toString());
39        notas.setNota(Double.parseDouble(etNota.getText().toString()));
40        banco.incluirRegistro(notas);
41        Toast.makeText(getApplicationContext(), "Registro inserido com sucesso!", Toast.LENGTH_SHORT).show();
42    }
43
44    public void btAlterarOnClick(View v) {
45        int id = Integer.parseInt(etCodigo.getText().toString());
46        Notas notas = new Notas();
47        notas.set_id(id);
48        notas.setNomeDisciplina(etNomeDisciplina.getText().toString());
49        notas.setNota(Double.parseDouble(etNota.getText().toString()));
50        banco.alterarRegistro(notas);
51        Toast.makeText(getApplicationContext(), "Registro alterado com sucesso!", Toast.LENGTH_SHORT).show();
52    }
53
54    public void btExcluirOnClick(View v) {
55        int id = Integer.parseInt(etCodigo.getText().toString());
56        banco.excluirRegistro(id);
57        Toast.makeText(getApplicationContext(), "Registro excluído com sucesso!", Toast.LENGTH_SHORT).show();
58    }
```

SQLITE OPEN HELPER

- Os métodos **btPesquisarOnClick()** e **btListarOnClick()** não sofreram alterações.
- O código abaixo é referente ao processo de realizar a pesquisa de notas.

```
81    protected void realizarPesquisa(int id) {
82        Notas notas = banco.pesquisarRegistro(id);
83
84        if(notas != null) {
85            etCodigo.setText(String.valueOf(notas.get_id()));
86            etNomeDisciplina.setText(notas.getNomeDisciplina());
87            etNota.setText(String.valueOf(notas.getNota()));
88
89            Toast.makeText(getApplicationContext(), "Registro encontrado!", Toast.LENGTH_SHORT).show();
90        } else {
91            Toast.makeText(getApplicationContext(), "Registro não encontado!", Toast.LENGTH_SHORT).show();
92        }
93    }
```

SQLITE OPEN HELPER

- O código para exibição dos registros é mostrado abaixo:

```java
package pm25s.aula12.usandosqlite;

import pm25s.aula12.bd.DatabaseHandler;
import android.app.Activity;
import android.database.Cursor;
import android.os.Bundle;
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;

public class ListaActivity extends Activity {

    private ListView lvRegistros;
    private DatabaseHandler banco;

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

        lvRegistros = (ListView) findViewById(R.id.lvListar);
        banco = new DatabaseHandler(this);
        montarListaRegistro();
    }
```

```java
    @SuppressWarnings("deprecation")
    private void montarListaRegistro() {
        Cursor registros = banco.listarRegistros();
        String nomeCamposTabela[] = new String[] {"nome_disciplina", "nota"};
        int nomeCamposTela[] = new int[] {android.R.id.text1, android.R.id.text2};
        SimpleCursorAdapter adapter = new SimpleCursorAdapter(getBaseContext(),
                                                              android.R.layout.two_line_list_item,
                                                              registros,
                                                              nomeCamposTabela,
                                                              nomeCamposTela);

        lvRegistros.setAdapter(adapter);
    }
}
```

SQLITE OPEN HELPER

- Com estas mudanças, alterou-se um pouco a estrutura do projeto, que passou a contar com mais classes, como pode ser observado na estrutura do nosso projeto na imagem abaixo: