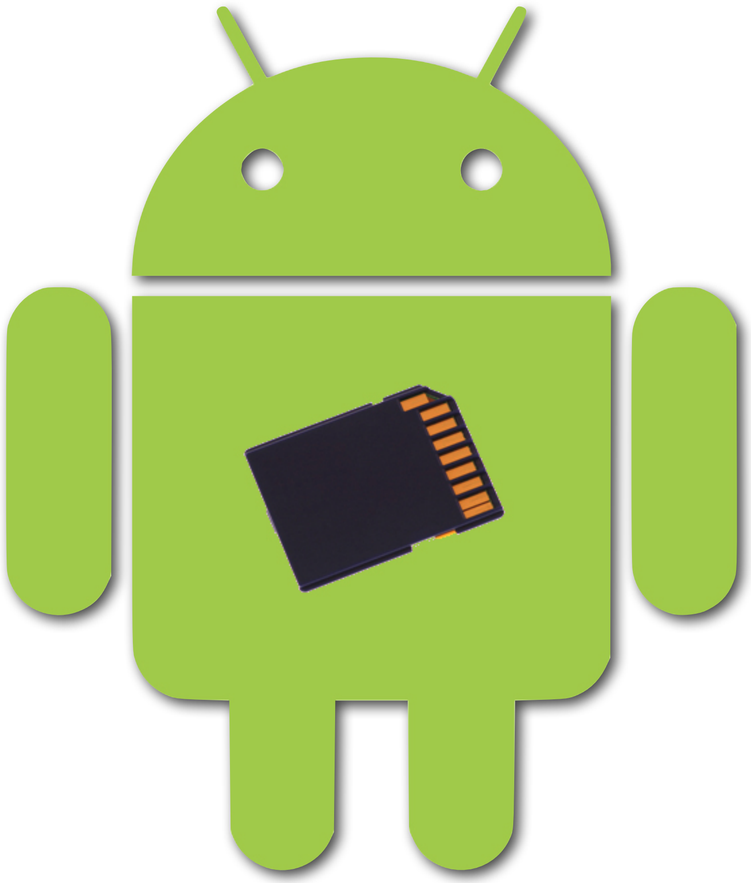
Entendendo as várias formas de persistir dados em Android

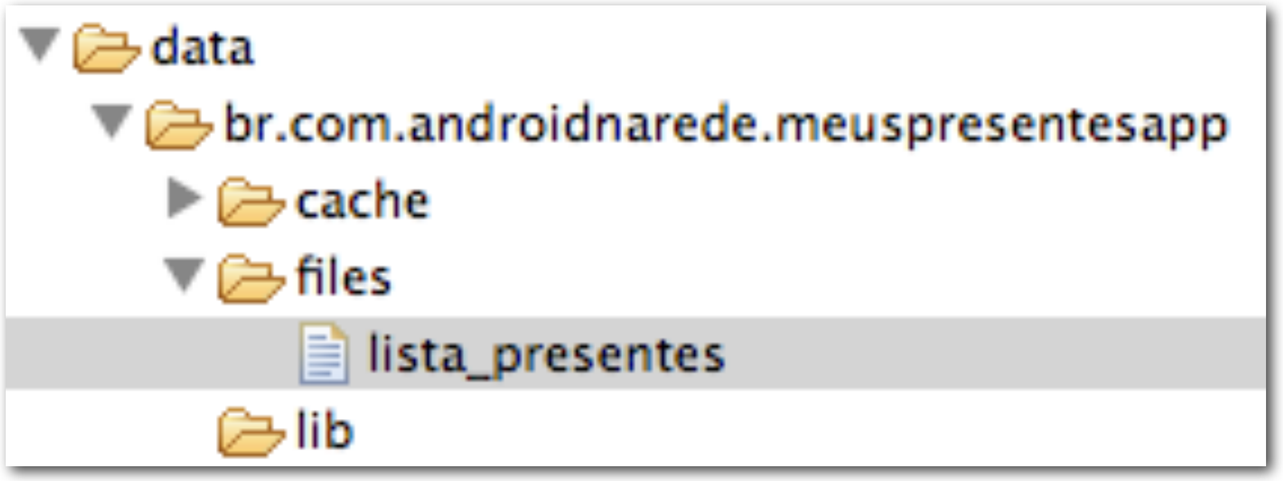
Formas de Persistência em Android

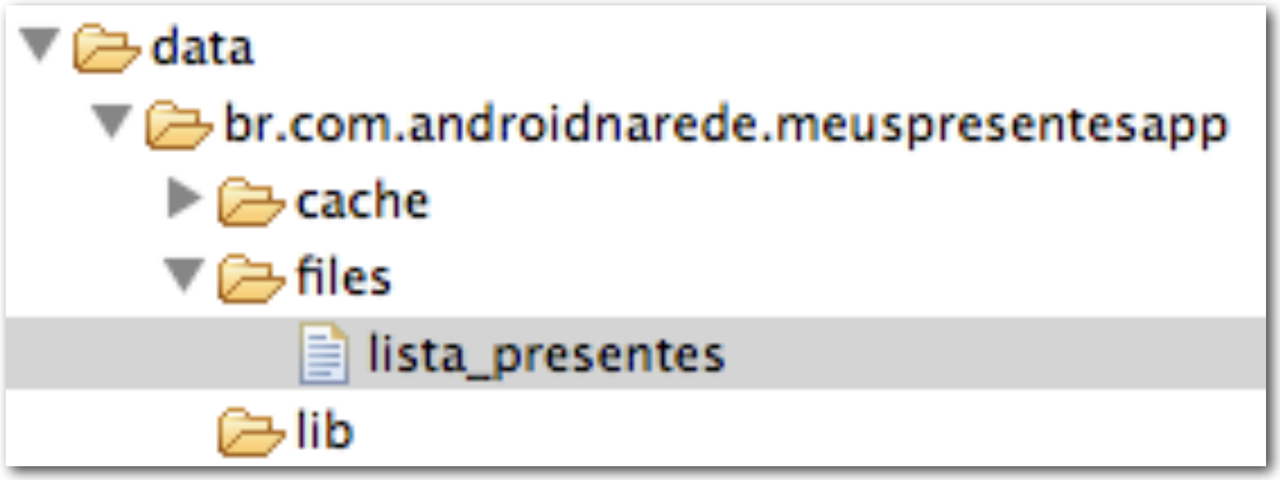
Formas de Persistência em Android



Em Android é possível armazenar seus dados de várias maneiras

Formas de Persistência em Android: Armazenamento Interno





Este tipo de armazenamento é **exclusivo** e **interno** à sua aplicação, **oculto** para **outras apps** e **usuário**

Armazenamento Interno: Gerando um arquivo de texto

```
public void exportar(List<Presente> listaPresentes) {
    try {
      StringBuilder linha = new StringBuilder();
      for (Presente presente : listaPresentes) {
        linha.append(presente.toString());
        linha.append("\n");
      }
      FileOutputStream fos = context.openFileOutput("lista_presentes.txt", Context.MODE_PRIVATE);
      fos.write(linha.toString().getBytes());
   } catch (FileNotFoundException e) {
      e.printStackTrace();
   } catch (IOException e) {
      e.printStackTrace();
   }
}
```

Para gerar o arquivo, basta chamar o método **Context.openFileOutput()**, informando o nome do arquivo e modo de leitura.

```
public void exportar(List<Presente> listaPresentes) {
    try {
      StringBuilder linha = new StringBuilder();
      for (Presente presente : listaPresentes) {
        linha.append(presente.toString());
        linha.append("\n");
      }
      FileOutputStream fos = context.openFileOutput("lista_presentes.txt", Context.MODE_PRIVATE);
      fos.write(linha.toString().getBytes());
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
}
```

Armazenamento Interno: Gerando um arquivo de texto

```
public void exportar(List<Presente> listaPresentes) {
    try {
      StringBuilder linha = new StringBuilder();
      for (Presente presente : listaPresentes) {
        linha.append(presente.toString());
        linha.append("\n");
      }
      FileOutputStream fos = context.openFileOutput("lista_presentes.txt", Context.MODE_PRIVATE);
      fos.write(linha.toString().getBytes());
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
```

Depois, basta chamarmos o método **FileOutputStream.write()** para finalizar a criação do arquivo.

```
public void exportar(List<Presente> listaPresentes) {
    try {
      StringBuilder linha = new StringBuilder();
      for (Presente presente : listaPresentes) {
        linha.append(presente.toString());
        linha.append("\n");
      }
      FileOutputStream fos = context.openFileOutput("lista_presentes.txt", Context.MODE_PRIVATE);
      fos.write(linha.toString().getBytes());
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
```

Armazenamento Interno: Lendo um arquivo interno

Para ler o arquivo gerado, chamamos o método **Context.openFileInput(),** informando o nome do arquivo que foi criado anteriormente.

```
InputStream is = openFileInput("lista_presentes.txt");
InputStreamReader isr = new InputStreamReader(is);
BufferedReader br = new BufferedReader(isr);
StringBuilder textoArquivo = new StringBuilder();
String linha = br.readLine();

while(linha != null){
    textoArquivo.append(linha);
    linha = br.readLine();
    textoArquivo.append(linha);
}

// configura o texto do arquivo, em um componente TextView
textView.setText(textoArquivo.toString());
```

Armazenamento Interno: Lendo um arquivo interno

Armazenamento Interno: Lendo um arquivo interno

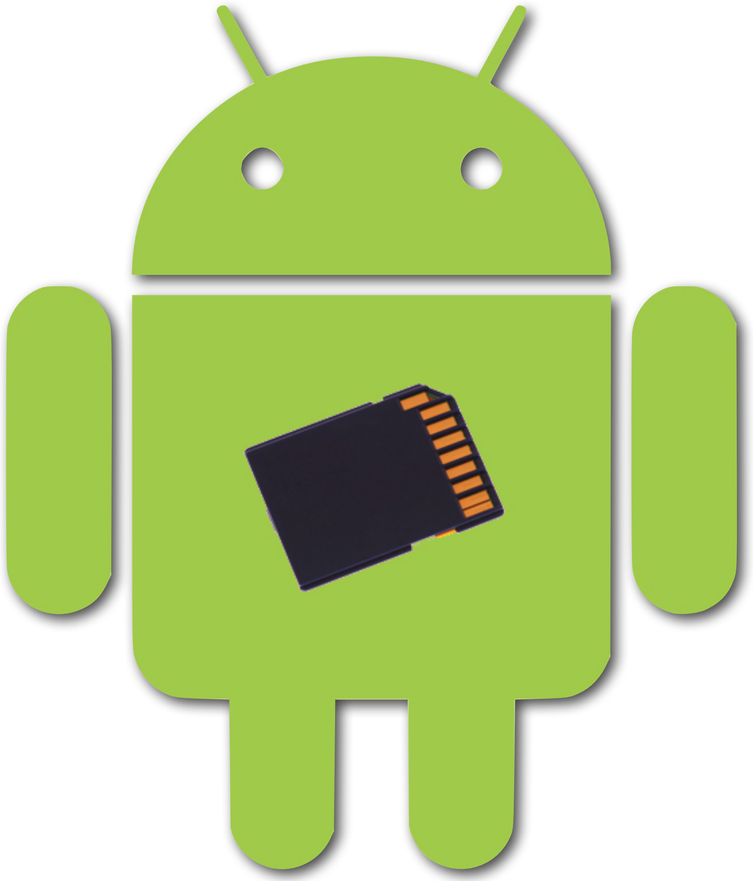
Feito isso, agora podemos utilizar as **classe utilitárias** para **leitura de arquivos** em Java.

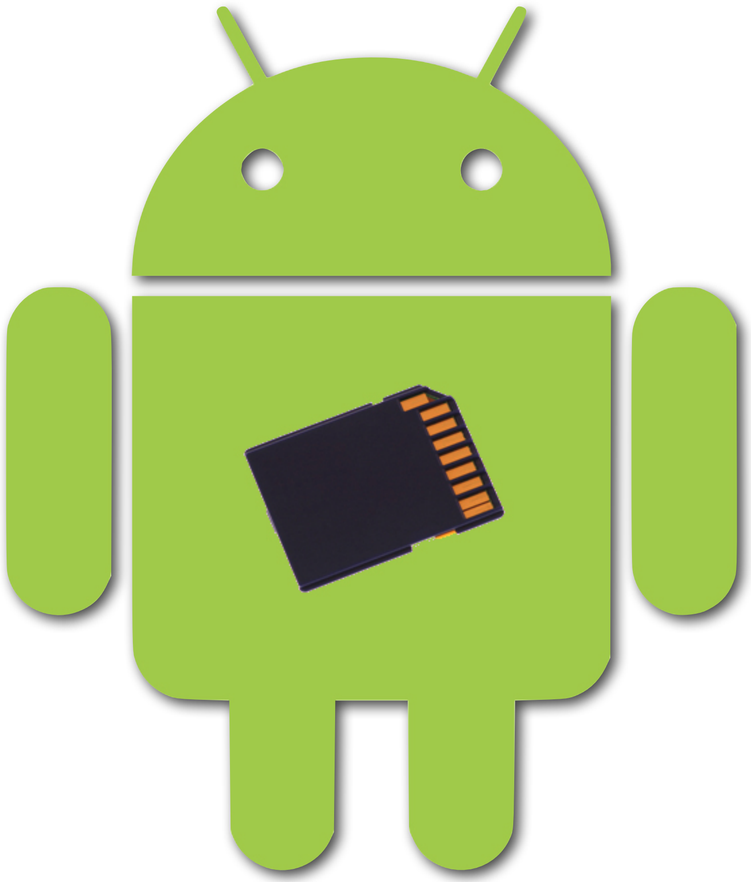
```
InputStream is = openFileInput("lista_presentes.txt");
InputStreamReader isr = new InputStreamReader(is);
BufferedReader br = new BufferedReader(isr);
StringBuilder textoArquivo = new StringBuilder();
String linha = br.readLine();

while(linha != null){
    textoArquivo.append(linha);
    linha = br.readLine();
    textoArquivo.append(linha);
}

// configura o texto do arquivo, em um componente TextView
textView.setText(textoArquivo.toString());
```

Formas de Persistência em Android: Armazenamento Externo





Este tipo de armazenamento é **visível** à **outras apps** e **usuário**

Armazenamento Externo: Gerando um arquivo externo

Armazenamento Externo: Gerando um arquivo externo

Utilizamos o método **Environment.getExternalStorageDirectory()** para referenciar ao diretório de armazenamento externo padrão no smartphone do usuário (ex: */mnt/sdcard*).

```
public void exportar(List<Presente> listaPresentes) {
    try {
       StringBuilder linha = new StringBuilder();
       for (Presente presente : listaPresentes) {
          linha.append(presente.toString());
          linha.append("\n");
       }
       FileOutputStream fos = new FileOutputStream(new
File(Environment.getExternalStorageDirectory() + File.separator + "lista_presentes.txt"));
       fos.write(linha.toString().getBytes());
    } catch (FileNotFoundException e) {
       e.printStackTrace();
    } catch (IOException e) {
       e.printStackTrace();
    }
  }
}
```

Armazenamento Externo: Lendo um arquivo externo

Armazenamento Externo: Lendo um arquivo externo

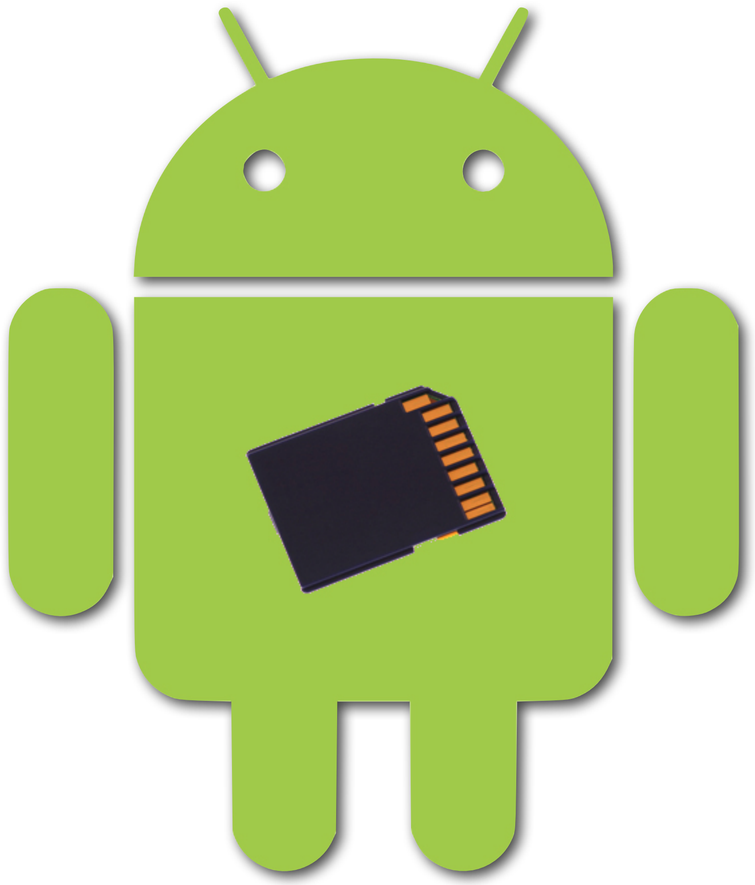
Feito isso, agora podemos utilizar as **classe utilitárias** para **leitura de arquivos** em Java.

```
File diretorioExterno = Environment.getExternalStorageDirectory();
File arquivo = new File(diretorioExterno + "/lista_presentes.txt");
FileInputStream fis = new FileInputStream(arquivo);
InputStreamReader isr = new InputStreamReader(fis);
BufferedReader br = new BufferedReader(isr);
StringBuilder textoArquivo = new StringBuilder();
String linha = br.readLine();

while(linha != null){
    textoArquivo.append(linha);
    linha = br.readLine();
    textoArquivo.append(linha);
}

// configura o texto do arquivo, em um componente TextView
textView.setText(textoArquivo.toString());
```

Formas de Persistência em Android: Shared Preferences



Preferências que são **compartilhadas** entre os componentes da app (ex: Activity), utilizadas para armazenar **dados primitivos** em pares de **chave-valor**

Shared Preferences: Armazenando preferências

```
SharedPreferences sharedPreferences =
context.getSharedPreferences("infos_usuario", Context.MODE_PRIVATE);

// utilizado para manipular os dados a serem persistidos
SharedPreferences.Editor editor = sharedPreferences.edit();
editor.putString("login", usuario.getLogin());
editor.putString("senha", usuario.getSenha());

// realiza as alterações
editor.commit();
```

1. Recuperamos uma referência para a classe **SharedPreferences**, de acordo com o método **Context.getSharedPreferences()**

```
SharedPreferences sharedPreferences =
context.getSharedPreferences("infos_usuario", Context.MODE_PRIVATE);

// utilizado para manipular os dados a serem persistidos
SharedPreferences.Editor editor = sharedPreferences.edit();
editor.putString("login", usuario.getLogin());
editor.putString("senha", usuario.getSenha());

// realiza as alterações
editor.commit();
```

nome da preferência

visibilidade da SharedPreferences

Shared Preferences: Armazenando preferências

```
SharedPreferences sharedPreferences =
context.getSharedPreferences("infos_usuario", Context.MODE_PRIVATE);

// utilizado para manipular os dados a serem persistidos
SharedPreferences.Editor editor = sharedPreferences.edit();
editor.putString("login", usuario.getLogin());
editor.putString("senha", usuario.getSenha());

// realiza as alterações
editor.commit();
```

2. Chamamos o método **SharedPreferences.edit()** para recuperar um objeto **SharedPreferences.Editor**

```
SharedPreferences sharedPreferences =
context.getSharedPreferences("infos_usuario", Context.MODE_PRIVATE);

// utilizado para manipular os dados a serem persistidos
SharedPreferences.Editor editor = sharedPreferences.edit();
editor.putString("login", usuario.getLogin());
editor.putString("senha", usuario.getSenha());

// realiza as alterações
editor.commit();
```

```
SharedPreferences sharedPreferences =
context.getSharedPreferences("infos_usuario", Context.MODE_PRIVATE);

// utilizado para manipular os dados a serem persistidos
SharedPreferences.Editor editor = sharedPreferences.edit();
editor.putString("login", usuario.getLogin());
editor.putString("senha", usuario.getSenha());

// realiza as alterações
editor.commit();
```

3. Agora armazenamos os dados da preferência, por meio do método **put<Tipo>(chave, valor)**, onde **<Tipo>** pode ser Boolean, Float, Int, Long ou String

```
SharedPreferences sharedPreferences =
context.getSharedPreferences("infos_usuario", Context.MODE_PRIVATE);

// utilizado para manipular os dados a serem persistidos
SharedPreferences.Editor editor = sharedPreferences.edit();
editor.putString("login", usuario.getLogin());
editor.putString("senha", usuario.getSenha());

// realiza as alterações
editor.commit();
```

```
SharedPreferences sharedPreferences =
context.getSharedPreferences("infos_usuario", Context.MODE_PRIVATE);

// utilizado para manipular os dados a serem persistidos
SharedPreferences.Editor editor = sharedPreferences.edit();
editor.putString("login", usuario.getLogin());
editor.putString("senha", usuario.getSenha());

// realiza as alterações
editor.commit();
```

4. Por fim, basta confirmarmos a mudança do editor, chamando o método **commit()**

```
SharedPreferences sharedPreferences =
context.getSharedPreferences("infos_usuario", Context.MODE_PRIVATE);

// utilizado para manipular os dados a serem persistidos
SharedPreferences.Editor editor = sharedPreferences.edit();
editor.putString("login", usuario.getLogin());
editor.putString("senha", usuario.getSenha());

// realiza as alterações
editor.commit();
```

Shared Preferences: Recuperando preferências

```
SharedPreferences sharedPreferences =
                    context.getSharedPreferences("infos_usuario",
                                                 Context.MODE_PRIVATE);
    String login = sharedPreferences.getString("login", null);
    String senha = sharedPreferences.getString("senha", null);
```

1. Para recuperarmos preferências, devemos chamarmos novamente o método **Context.getSharedPreferences()**

```
SharedPreferences sharedPreferences =
                    context.getSharedPreferences("infos_usuario",
                                                 Context.MODE_PRIVATE);
String login = sharedPreferences.getString("login", null);
String senha = sharedPreferences.getString("senha", null);
```

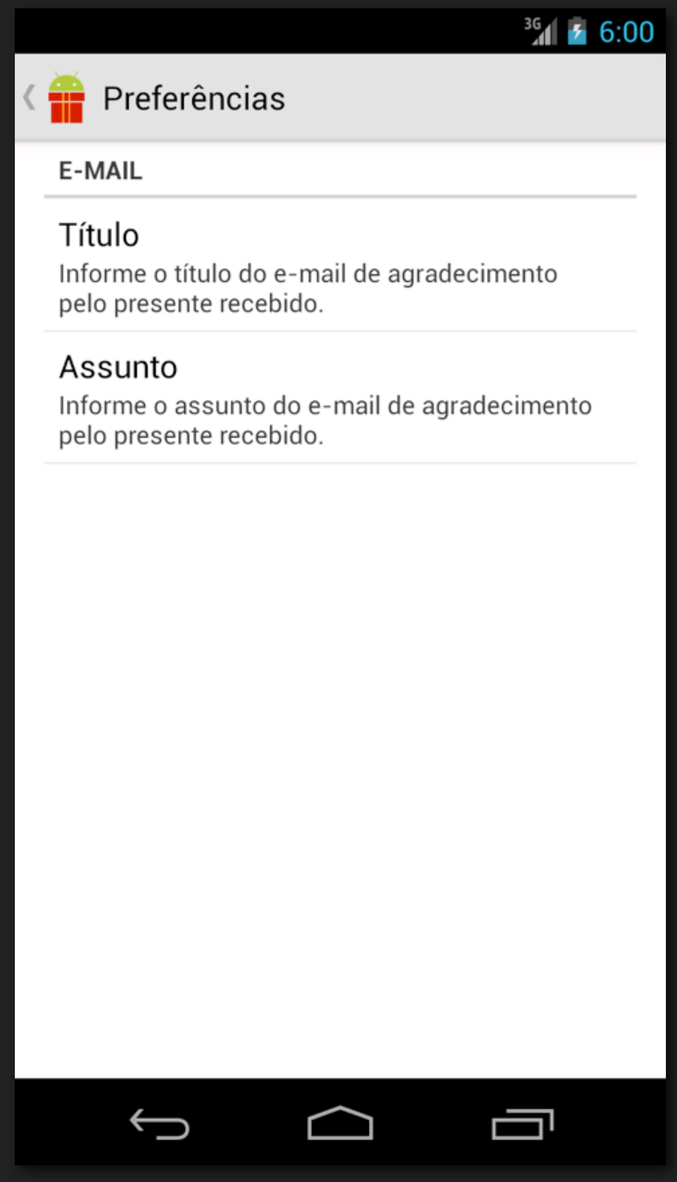
Shared Preferences: Recuperando preferências

```
SharedPreferences sharedPreferences =
                    context.getSharedPreferences("infos_usuario",
                                                 Context.MODE_PRIVATE);
    String login = sharedPreferences.getString("login", null);
    String senha = sharedPreferences.getString("senha", null);
```

2. Recupere o valor armazenado, chamando o método **SharedPreferences.get<Tipo>**

```
SharedPreferences sharedPreferences =
                    context.getSharedPreferences("infos_usuario",
                                                 Context.MODE_PRIVATE);
    String login = sharedPreferences.getString("login", null);
    String senha = sharedPreferences.getString("senha", null);
```

Entendendo como implementar uma camada de preferências para sua app

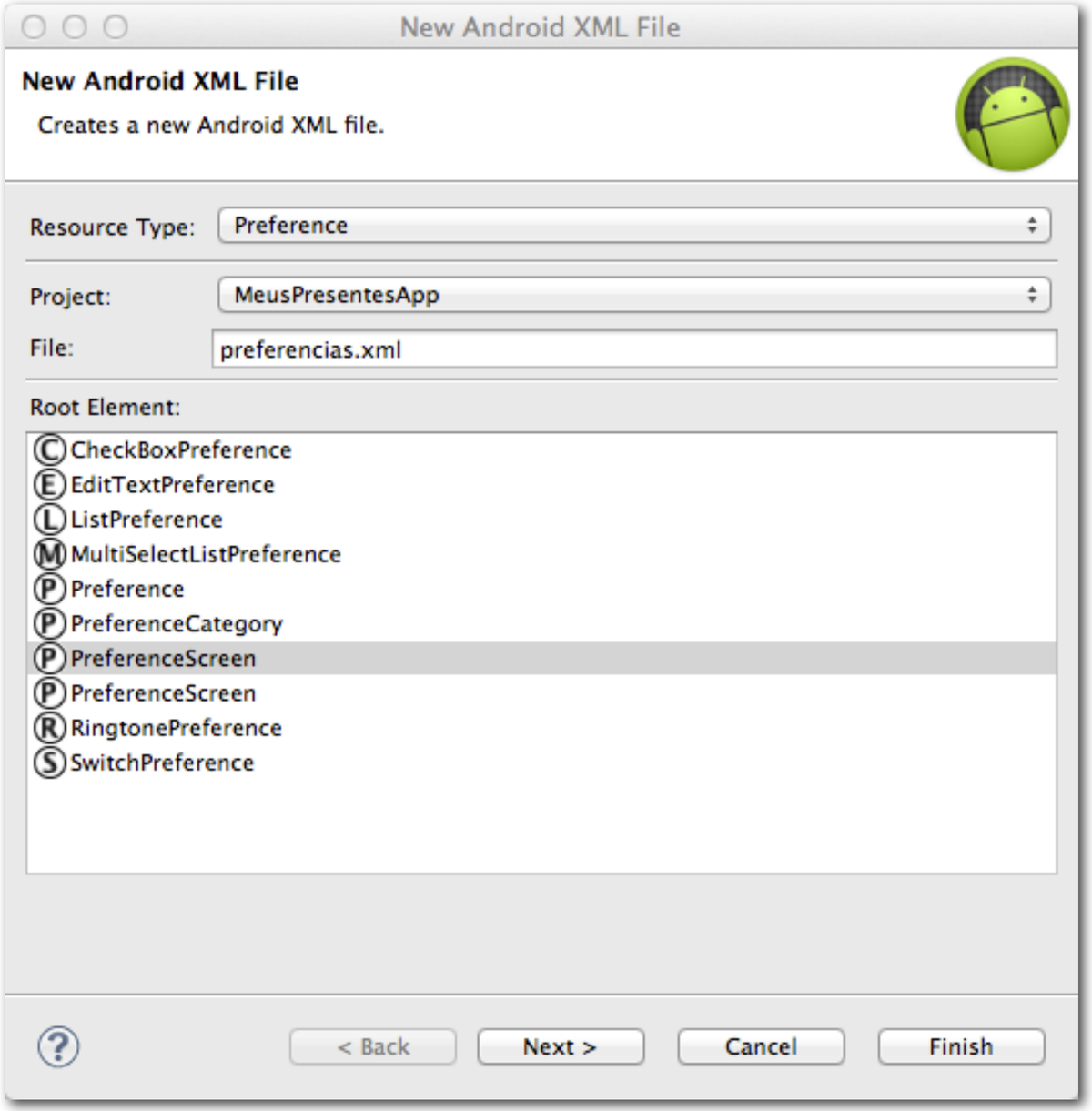


API de Preferências

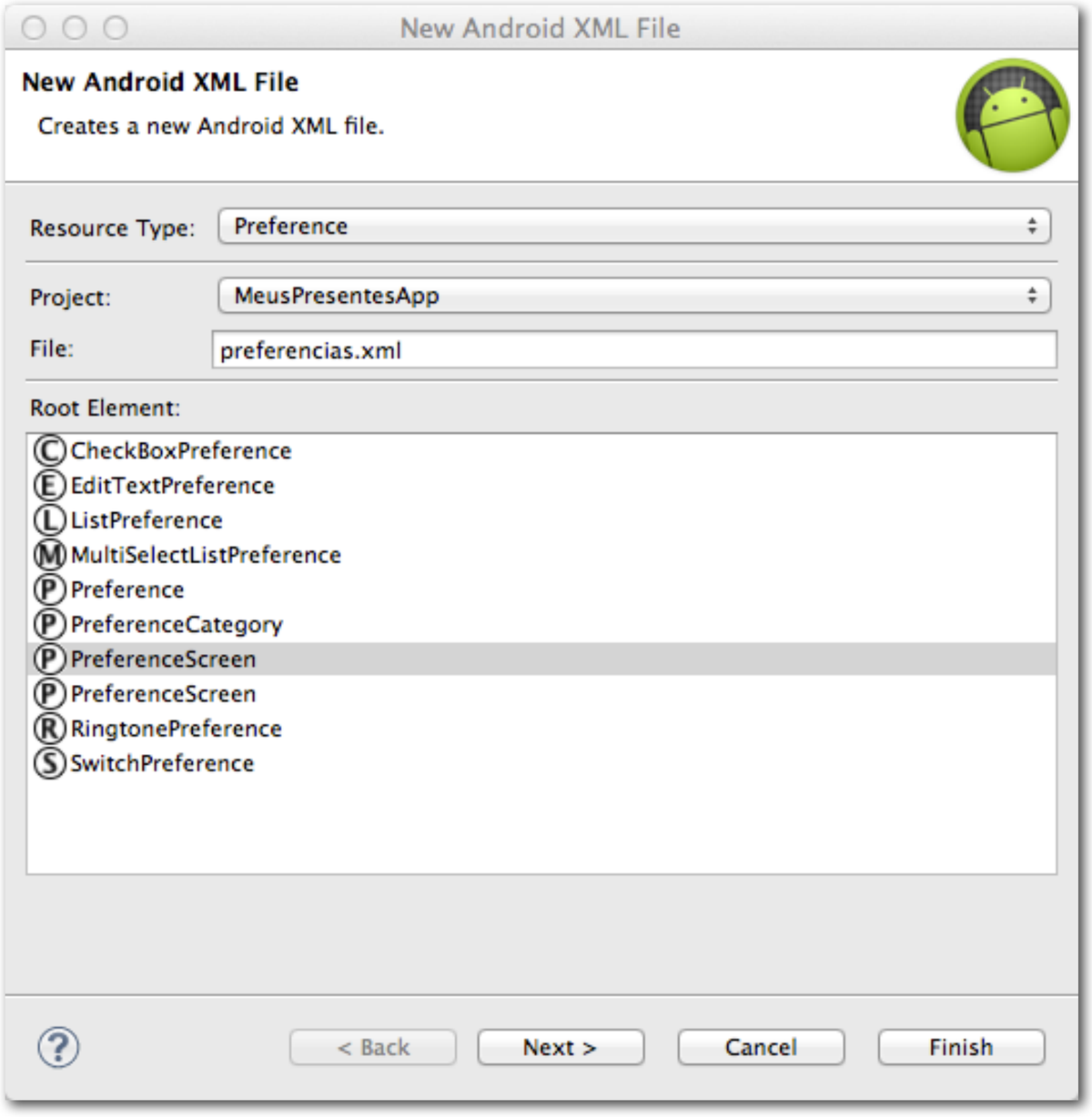
API de Preferências

Preferência é uma maneira de deixar os **usuários personalizarem** as **apps** como desejarem.

API de Preferências: Criando um arquivo de preferências



API de Preferências: Criando um arquivo de preferências



As preferências podem ser criadas facilmente indo em ***File > New > Android XML File*** e em ***Resource Type***, escolha *Preference*.

API de Preferências: Definindo preferências em XML

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
        <PreferenceCategory android:title="E-mail">
                <EditTextPreference
                        android:title="Título"
                        android:defaultValue="Obrigado pelo Presente"
                        android:summary="Informe o título do e-mail de
agradecimento pelo presente recebido."
                        android:key="preferencia_titulo_email" />
                <EditTextPreference
                        android:title="Assunto"
                        android:defaultValue="Muito obrigado pelo presente
recebido."
                        android:summary="Informe o assunto do e-mail de
agradecimento pelo presente recebido."
                        android:key="preferencia_assunto_email" />
        </PreferenceCategory>

        ...
</PreferenceScreen>
```

API de Preferências: Definindo preferências em XML

As preferências podem ser definidas totalmente em XML, localizadas na pasta **res/xml**

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
        <PreferenceCategory android:title="E-mail">
                <EditTextPreference
                        android:title="Título"
                        android:defaultValue="Obrigado pelo Presente"
                        android:summary="Informe o título do e-mail de
agradecimento pelo presente recebido."
                        android:key="preferencia_titulo_email" />
                <EditTextPreference
                        android:title="Assunto"
                        android:defaultValue="Muito obrigado pelo presente
recebido."
                        android:summary="Informe o assunto do e-mail de
agradecimento pelo presente recebido."
                        android:key="preferencia_assunto_email" />
        </PreferenceCategory>

        ...
</PreferenceScreen>
```

API de Preferências: Utilizando a PreferenceActivity

```
public class PreferenciasActivity extends PreferenceActivity {

 /** Called when the activity is first created. */
 @Override
 public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   addPreferencesFromResource(R.xml.preferencias);
 }
}
```

Depois disso, criamos uma Activity que estenda **PreferenceActivity**.

```
public class PreferenciasActivity extends PreferenceActivity {

 /** Called when the activity is first created. */
 @Override
 public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   addPreferencesFromResource(R.xml.preferencias);
 }
}
```

API de Preferências: Utilizando a PreferenceActivity

```
public class PreferenciasActivity extends PreferenceActivity {

  private EditTextPreference preferenciaTituloEmail;
  private EditTextPreference preferenciaAssuntoEmail;


  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    addPreferencesFromResource(R.xml.preferencias);
  }
}
```

Recuperamos, **em Java**, os componentes de preferência definidos em **XML**

```
public class PreferenciasActivity extends PreferenceActivity {

  private EditTextPreference preferenciaTituloEmail;
  private EditTextPreference preferenciaAssuntoEmail;


  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    addPreferencesFromResource(R.xml.preferencias);
  }
}
```

API de Preferências: Utilizando a PreferenceActivity

```
public class PreferenciasActivity extends PreferenceActivity {

  private EditTextPreference preferenciaTituloEmail;
  private EditTextPreference preferenciaAssuntoEmail;


  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    addPreferencesFromResource(R.xml.preferencias);
  }
}
```

Depois, chamamos o método **addPreferencesFromResource()**, referenciando o xml de preferencias.

```
public class PreferenciasActivity extends PreferenceActivity {

  private EditTextPreference preferenciaTituloEmail;
  private EditTextPreference preferenciaAssuntoEmail;


  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    addPreferencesFromResource(R.xml.preferencias);
  }
}
```

API de Preferências: Persistindo preferências

```
// recupera as preferências do arquivo res/xml/preferencias.xml
preferenciaTituloEmail = (EditTextPreference) findPreference("preferencia_titulo_email");
preferenciaAssuntoEmail = (EditTextPreference) findPreference("preferencia_assunto_email");
SharedPreferences preferencias = getSharedPreferences("preferencias", Context.MODE_PRIVATE);
SharedPreferences.Editor editor = preferencias.edit();
editor.putString("pref_titulo_email", preferenciaTituloEmail.getText().toString());
editor.putString("pref_assunto_email", preferenciaAssuntoEmail.getText().toString());
editor.commit();
```

API de Preferências: Carregando preferências

```
// recupera as preferências do arquivo res/xml/preferencias.xml
preferenciaTituloEmail = (EditTextPreference) findPreference("preferencia_titulo_email");
preferenciaAssuntoEmail = (EditTextPreference) findPreference("preferencia_assunto_email");
SharedPreferences preferencias = getSharedPreferences("preferencias", Context.MODE_PRIVATE);
String prefTituloEmail = preferencias.getString("pref_titulo_email", “”);
String prefAssuntoEmail = preferencias.getString("pref_assunto_email", “”);
```

Mão na Massa!

1) Crie uma app que permita criar um arquivo internamente à app.

2) Crie uma app que permita criar um arquivo externamente à app.

3) Crie uma app e defina um arquivo XML de preferências para carregar em uma Activity de preferências.

4) Crie uma app que utilize SharedPreferences para salvar e carregar as preferências do usuário.

Aprendendo como trabalhar com banco de dados em Android


SQLite

Formas de Persistência em Android: Banco de Dados SQLite

Segue o padrão
SQL ANSI 92

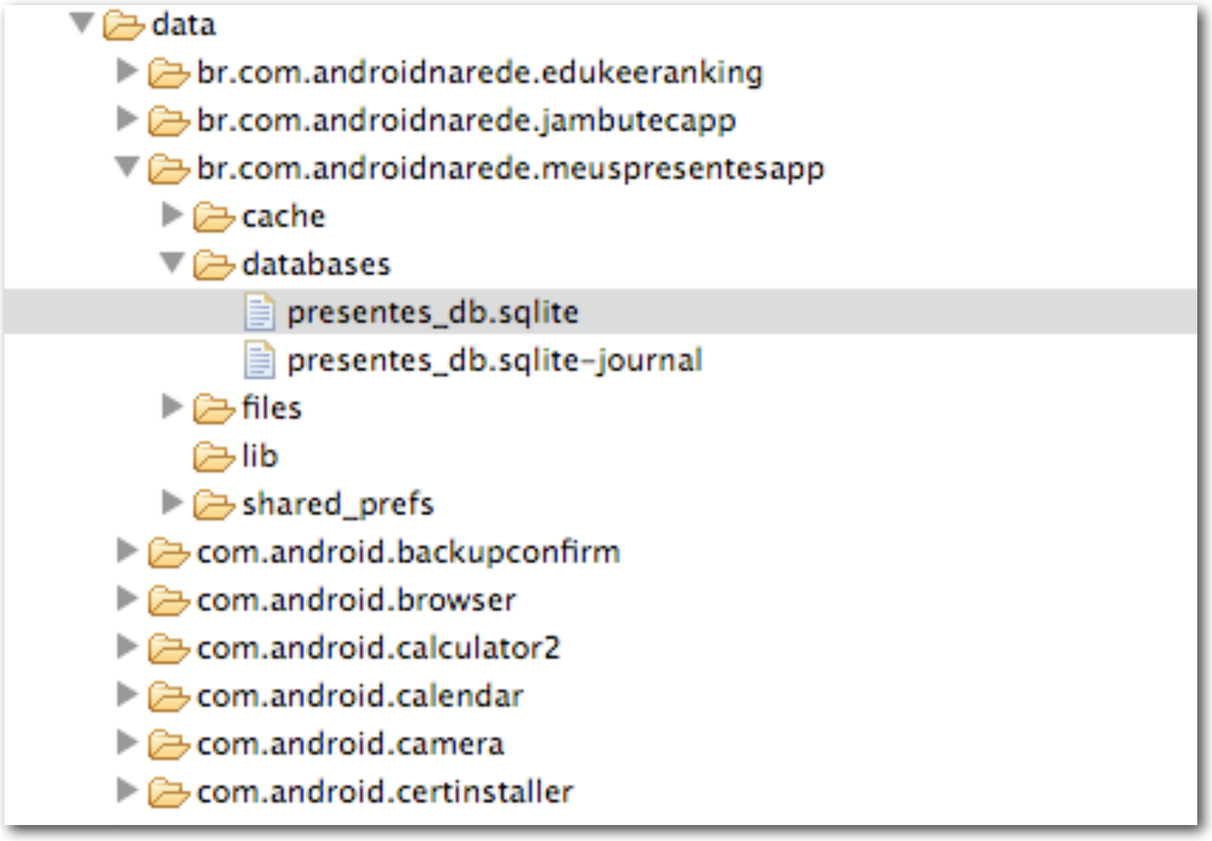
Android vem
com o **SQLite 3**

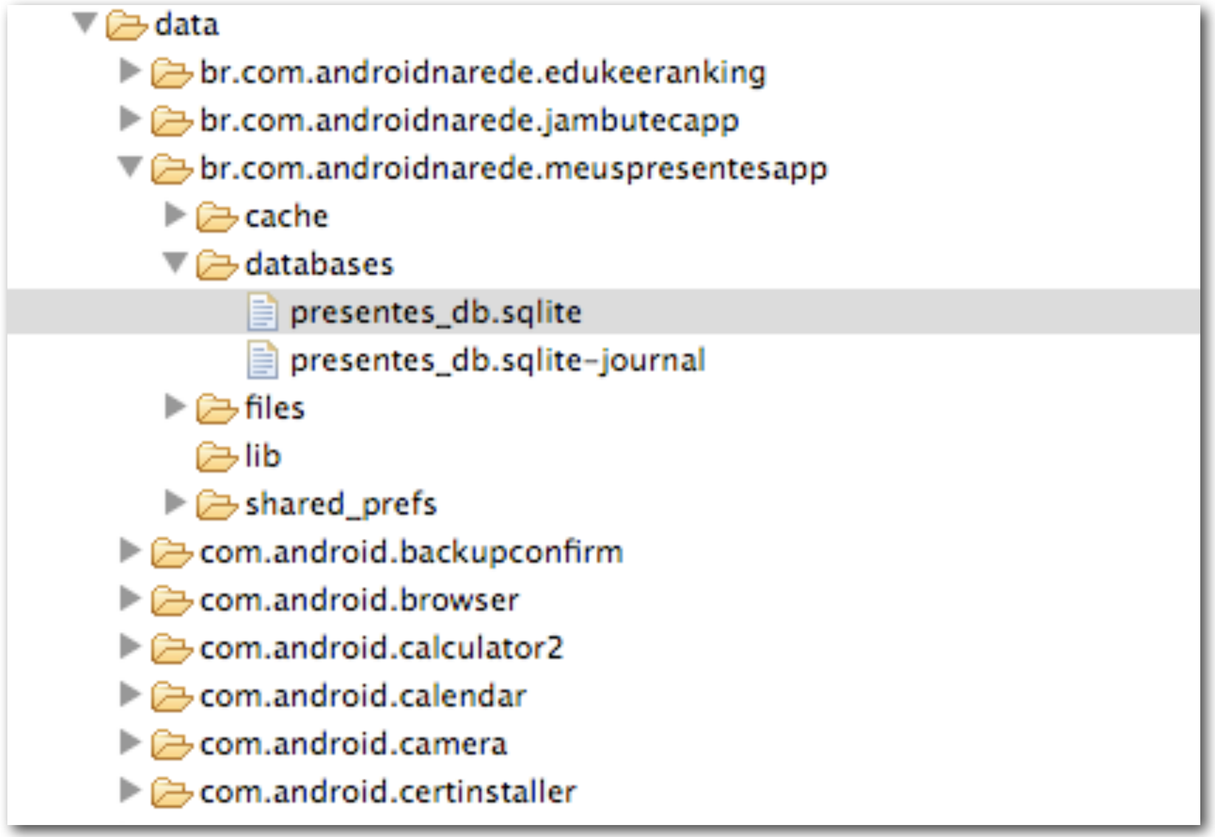
É um **banco** de
dados **relacional**
embarcado e
rápido

Suporta os
seguintes **tipos**:
INTEGER
PRIMARY KEY,
TEXT, NUMERIC,
BLOB

permite realizar
procedimentos de
consulta, **criação**
e **manipulação** em
tabelas

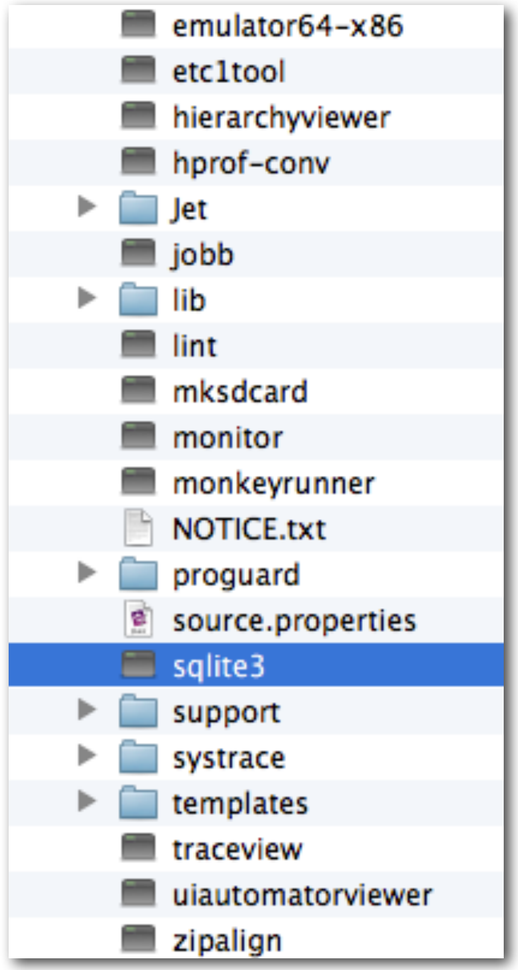
sqlite.org


data
br.com.androidnarede.edukeeranking
br.com.androidnarede.jambutecapp
br.com.androidnarede.meuspresentesapp
cache
databases
presentes_db.sqlite
presentes_db.sqlite-journal
files
lib
shared_prefs
com.android.backupconfirm
com.android.browser
com.android.calculator2
com.android.calendar
com.android.camera
com.android.certinstaller

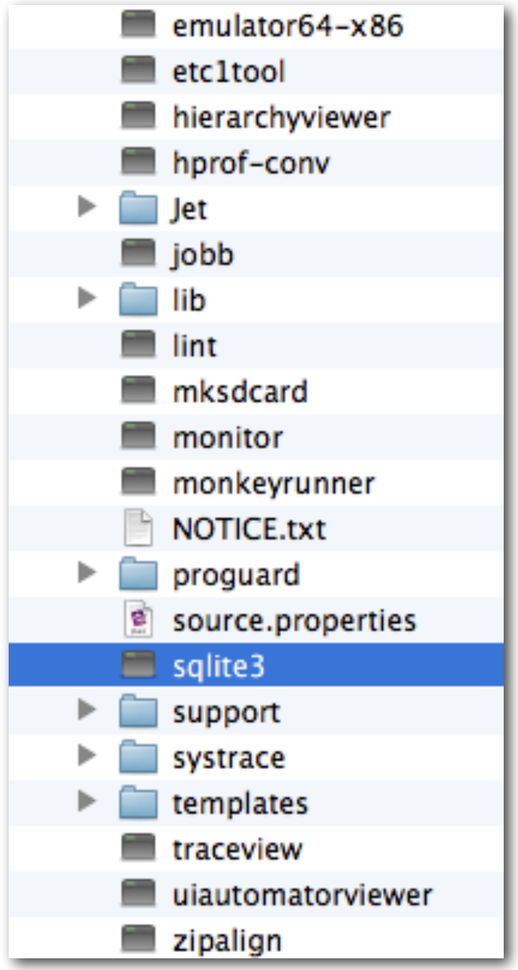


Todo banco de dados é interno à app, localizado em **/data/data/{pacote.da.app}/databases/**

Conhecendo a ferramenta sqlite3



Conhecendo a ferramenta sqlite3



diretório:
<ANDROID_HOME>/tools/sqlite3

Cliente em linha de comando que já vem no SDK. Permite **visualizar** conteúdo das **tabelas**, executar **comandos SQL** e realizar **outras funções** em banco de dados **SQLite**

Conectando com um banco de dados remoto com sqlite3

```
$ adb -s emulator-5554 shell
# sqlite3 /data/data/com.example.google.rss.rssexample/databases/rssitems.db
SQLite version 3.3.12
Enter ".help" for instructions
.... enter commands, then quit...
sqlite> .exit
```

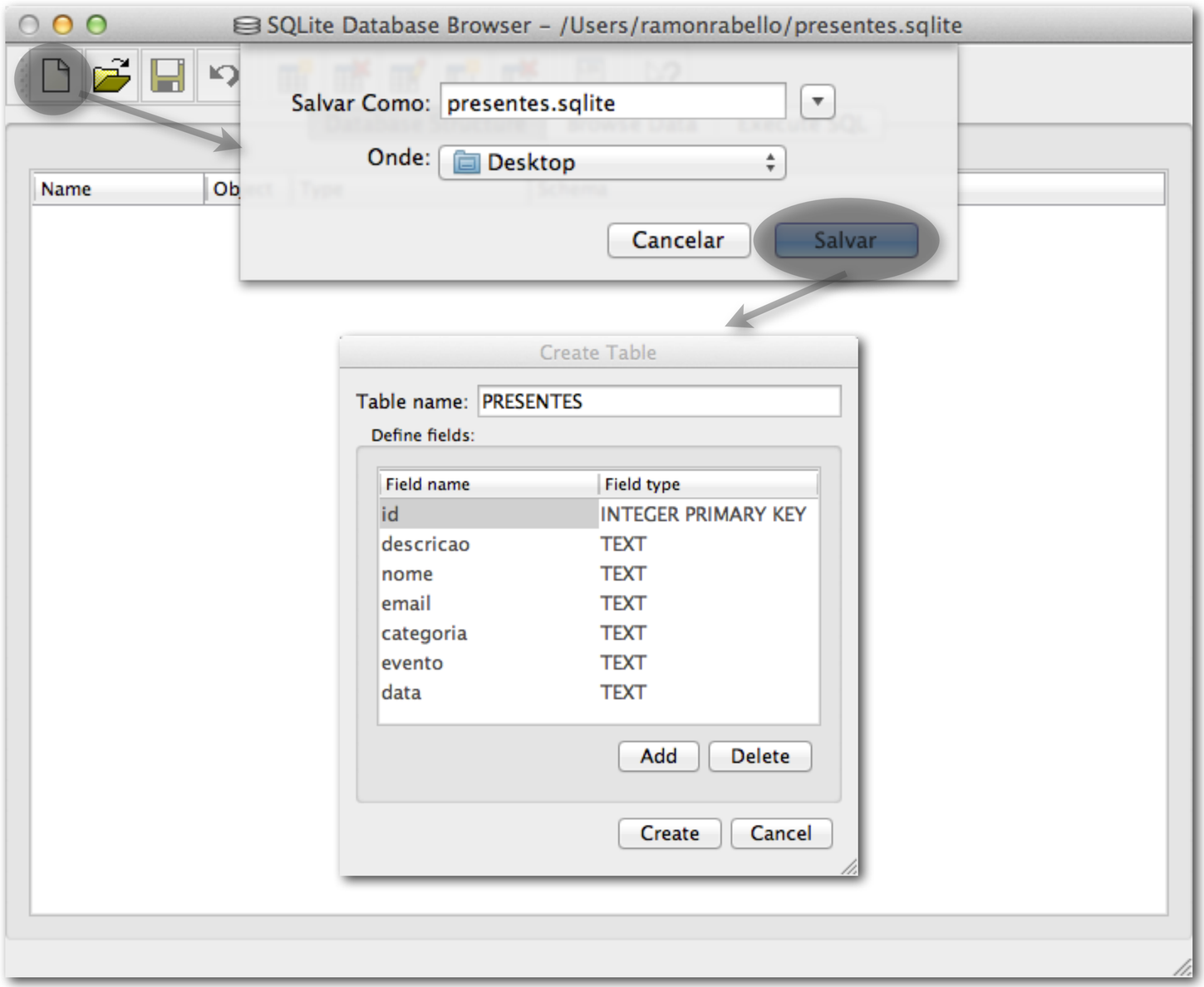
Conectando com um banco de dados remoto com sqlite3

```
$ adb -s emulator-5554 shell
# sqlite3 /data/data/com.example.google.rss.rssexample/databases/rssitems.db
SQLite version 3.3.12
Enter ".help" for instructions
.... enter commands, then quit...
sqlite> .exit
```

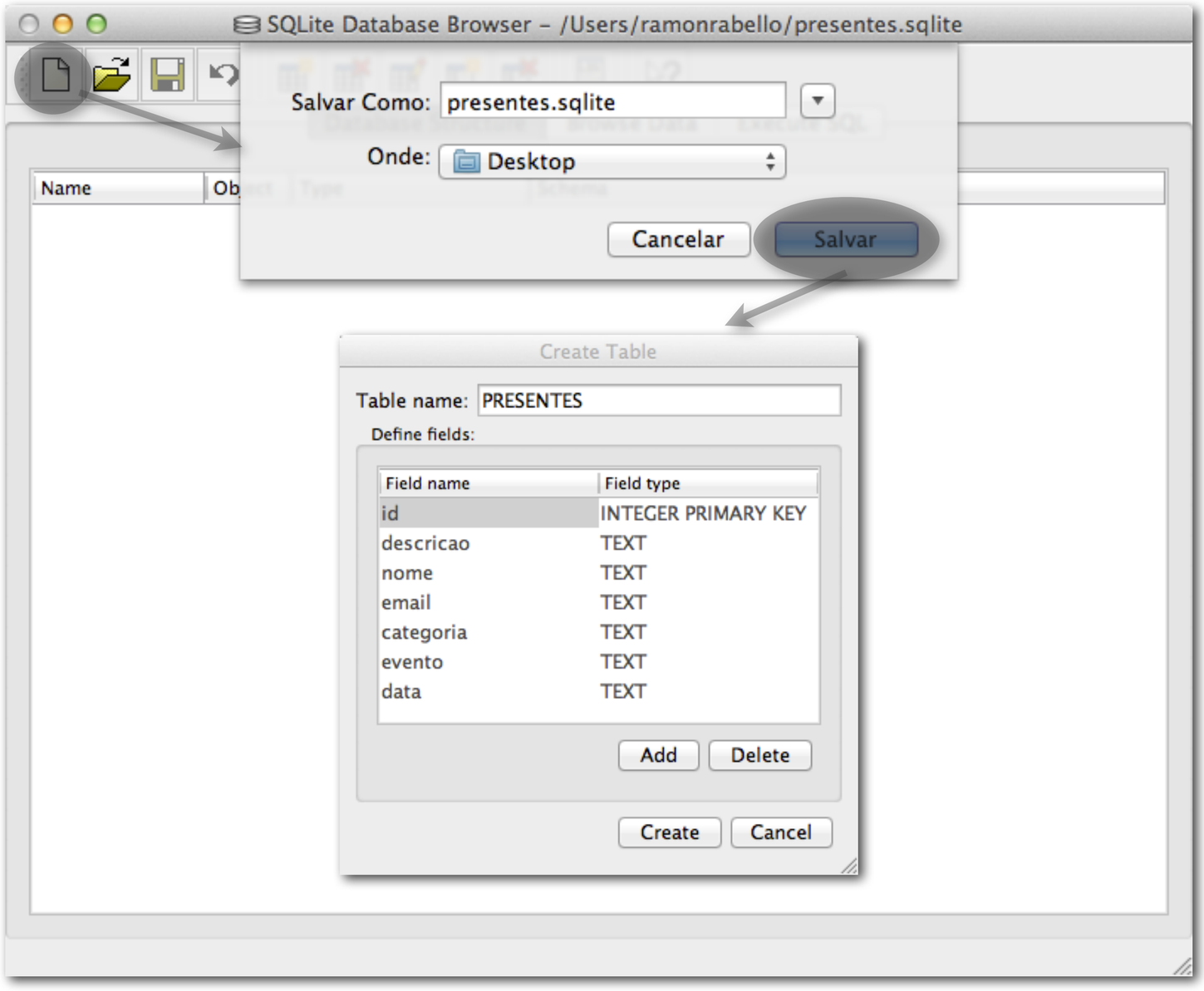
Depois de conectar com o banco de dados remoto, basta realizar alguma operação SQL:

Ex: ***SELECT * FROM <TABELA>***

Criando um banco de dados: Utilizando um front-end



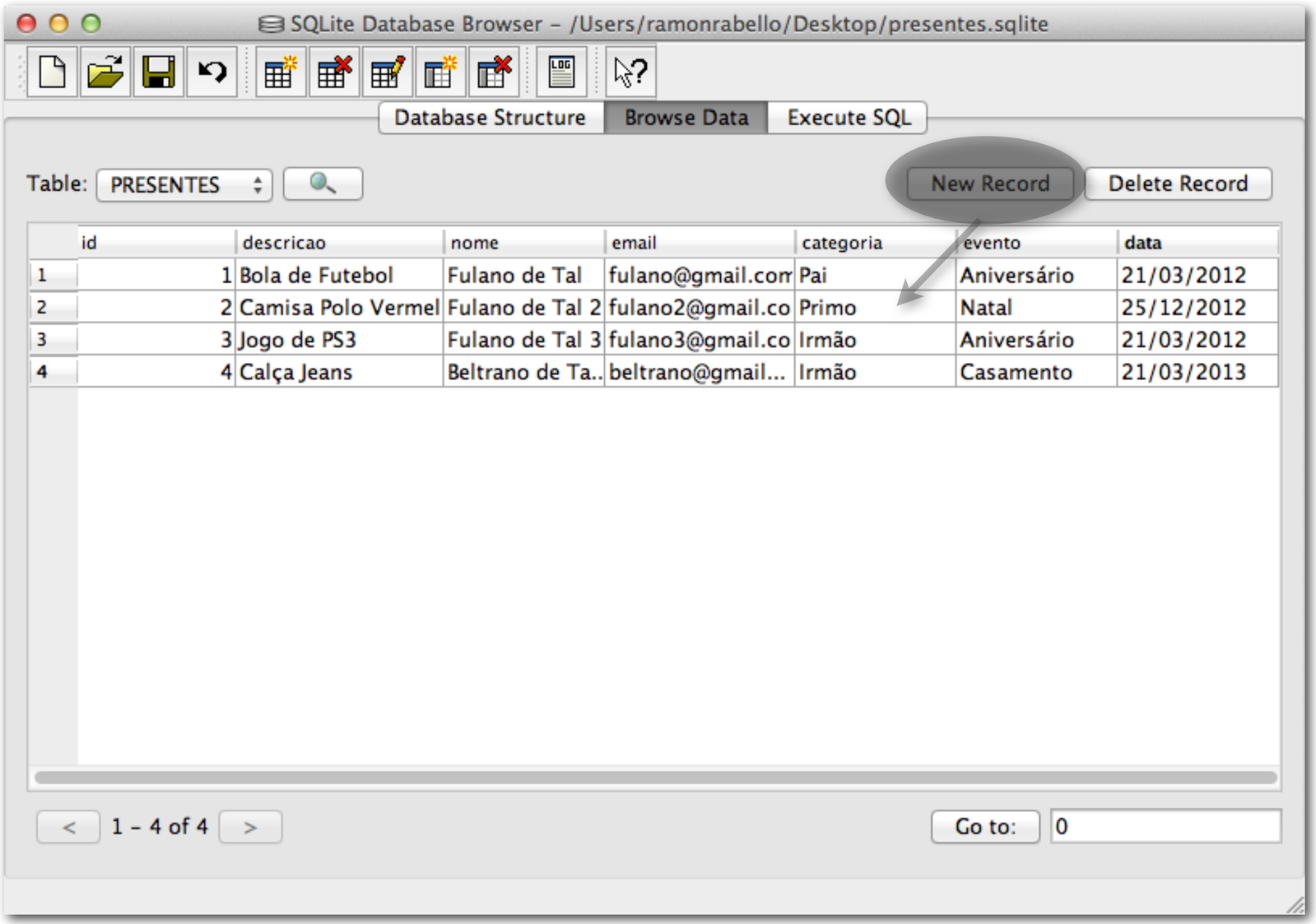
http://sourceforge.net/projects/sqlitebrowser/

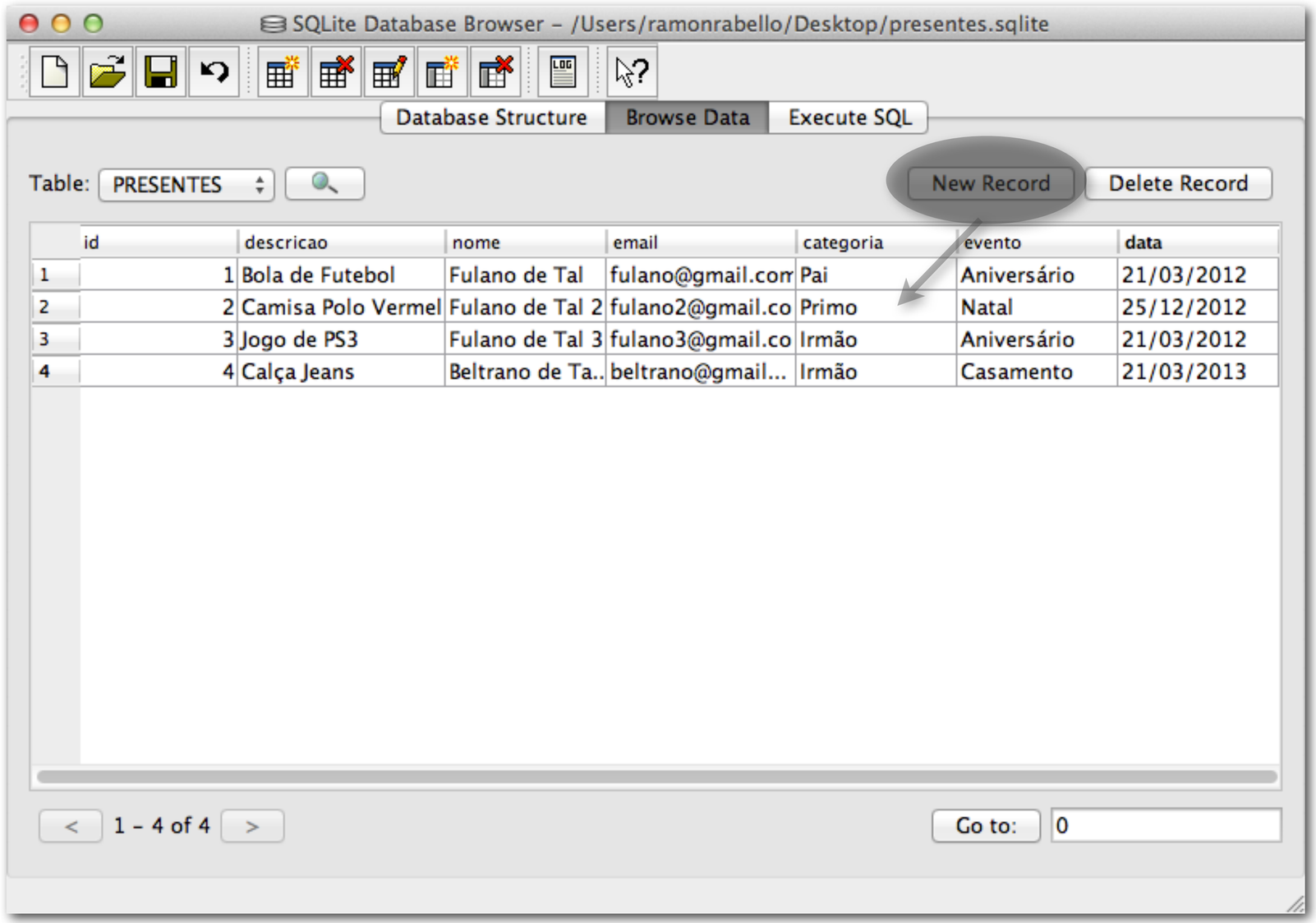


SQLite Database Browser

http://sourceforge.net/projects/sqlitebrowser/

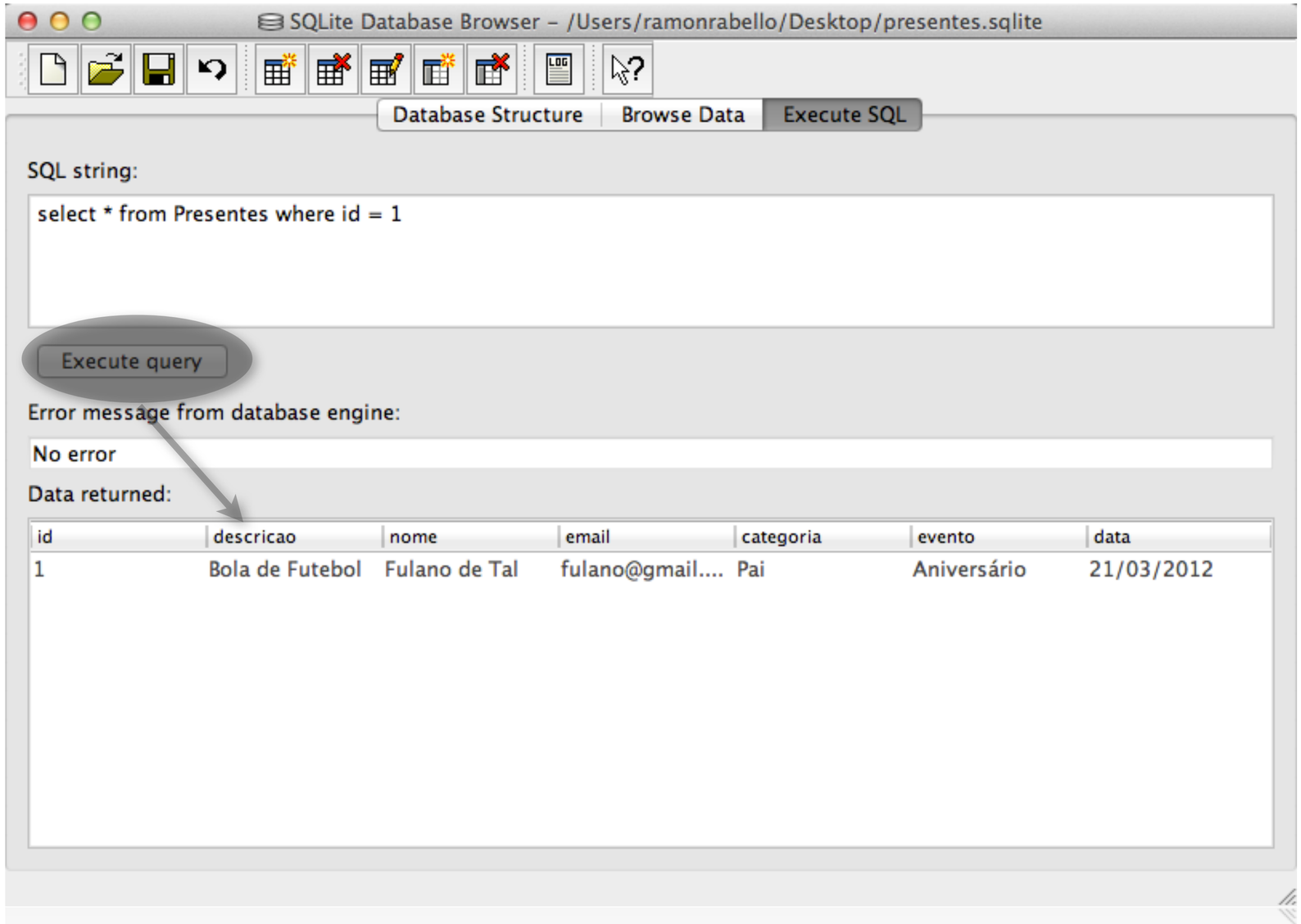
Gerenciando um banco de dados: Utilizando um front-end

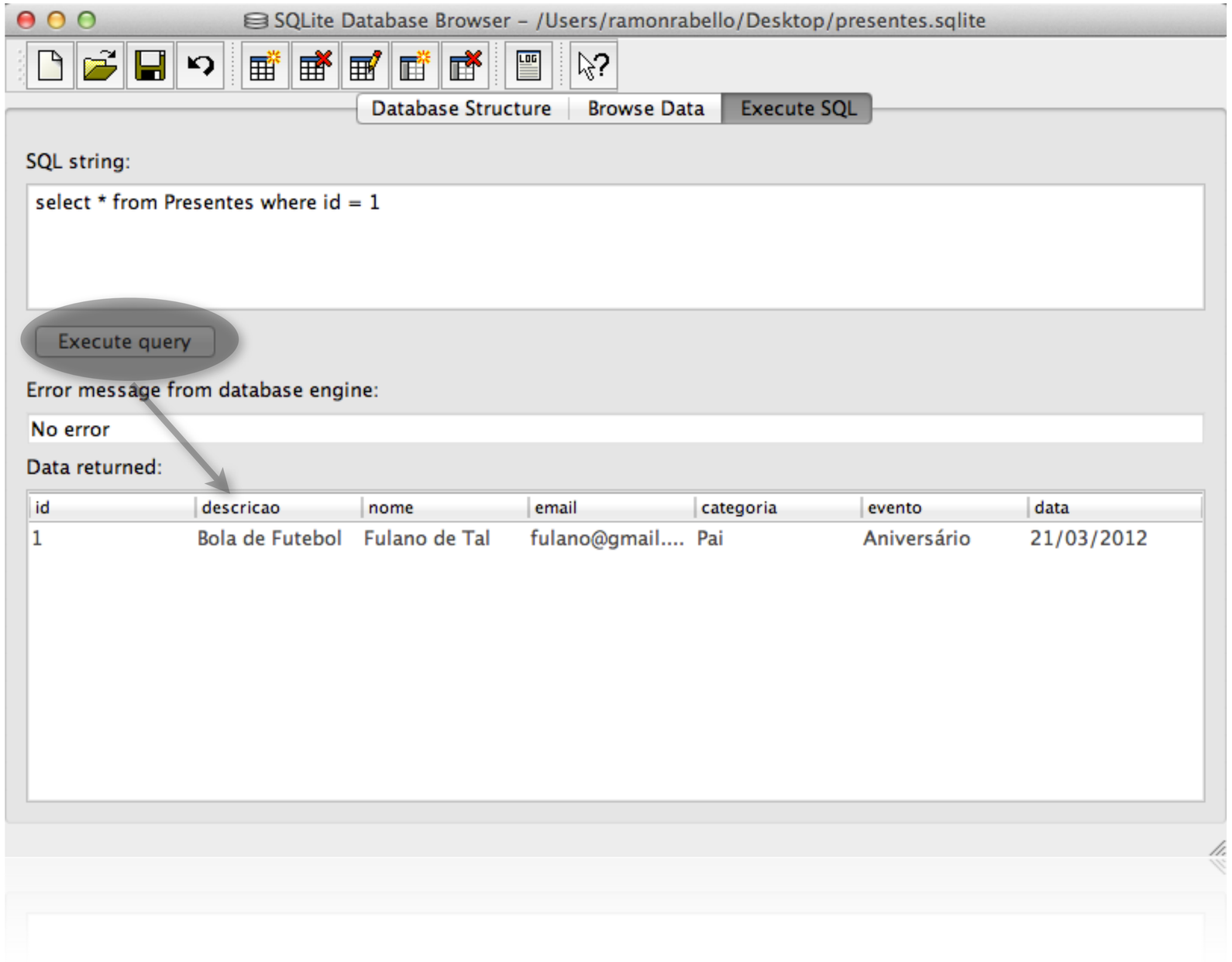




Gerenciando Registros

Consultando um banco de dados: Utilizando um front-end





Consultando Registros

Realizando conexão com o banco de dados

```
public SQLiteDatabase conectar(Context contexto, String nomeBanco){
    if (conexao == null){
        conexao = contexto.openOrCreateDatabase(nomeBanco, Context.MODE_PRIVATE, null);
    }
    return conexaoSqlite;
 }
```

A classe **SQLiteDatabase** é a principal que irá comunicar com o banco de dados, permitindo operações de criação, consulta ou manipulação de dados.

```
public SQLiteDatabase conectar(Context contexto, String nomeBanco){
    if (conexao == null){
        conexao = contexto.openOrCreateDatabase(nomeBanco, Context.MODE_PRIVATE, null);
    }
    return conexaoSqlite;
 }
```

Realizando conexão com o banco de dados

```
public SQLiteDatabase conectar(Context contexto, String nomeBanco){
    if (conexao == null){
        conexao = contexto.openOrCreateDatabase(nomeBanco, Context.MODE_PRIVATE, null);
    }
    return conexaoSqlite;
 }
```

Para iniciar uma conexão, basta chamar o método **Context.openOrCreateDatabase()**, informando o nome e a visibilidade.

```
public SQLiteDatabase conectar(Context contexto, String nomeBanco){
    if (conexao == null){
        conexao = contexto.openOrCreateDatabase(nomeBanco, Context.MODE_PRIVATE, null);
    }
    return conexaoSqlite;
 }
```

Criando tabelas: Dinamicamente em código-fonte

```
public void abreOuCriaBancoDeDados() {
    try {
      bancoDeDadosDePessoa = contexto.openOrCreateDatabase(NOME_BD,
Context.MODE_PRIVATE, null);
      bancoDeDadosDaAgenda.execSQL(PESSOA_CREATE_TABLE_DDL);
    } catch (Exception erro) {
      Toast.makeText(this, "Erro ao abrir banco de dados!", Toast.LENGTH_SHORT).show();
    }
  }
```

Depois disso, basta chamar o método **SQLiteDatabase.execSQL()** informando o script de criação de tabelas

```
public void abreOuCriaBancoDeDados() {
    try {
      bancoDeDadosDePessoa = contexto.openOrCreateDatabase(NOME_BD,
Context.MODE_PRIVATE, null);
      bancoDeDadosDaAgenda.execSQL(PESSOA_CREATE_TABLE_DDL);
    } catch (Exception erro) {
      Toast.makeText(this, "Erro ao abrir banco de dados!", Toast.LENGTH_SHORT).show();
    }
  }
```

Consultando registros no banco de dados

```
@Override
  public List<Presente> obterTodos() {
     ArrayList<Presente> presentes = new ArrayList<Presente>();
     Cursor c = conexaoUtil.getConexaoSqlite().query("presentes",

                                                     new String[]{ "id",
                                                                   "descricao",
                                                                   "nome",
                                                                   "categoria" },

                                                      null,
                                                      null,
                                                      null,
                                                      null,
                                                      null
                                                    );
}
```

nome da tabela

nome das colunas (projeção)

seleção

argumentos da seleção

groupBy

having

orderBy

Consultando registros no banco de dados

Para consultar, basta chamarmos o método **SQLiteDatabase.query()**, informando os parâmetros de filtro.

```
@Override
  public List<Presente> obterTodos() {
     ArrayList<Presente> presentes = new ArrayList<Presente>();
     Cursor c = conexaoUtil.getConexaoSqlite().query("presentes",

                                                     new String[]{ "id",
                                                                   "descricao",
                                                                   "nome",
                                                                   "categoria" },

                                                      null,
                                                      null,
                                                      null,
                                                      null,
                                                      null
                                                    );
}
```

nome da tabela

nome das colunas (projeção)

seleção

argumentos da seleção

groupBy

having

orderBy

Consultando registros no banco de dados

```
@Override
   public List<Presente> obterTodos() {
      ArrayList<Presente> presentes = new ArrayList<Presente>();
      Cursor c = conexaoUtil.getConexaoSqlite().query(...);

      while (c.moveToNext()) {
         Integer id = c.getInt(0);
         String descricao = c.getString(1);
         String nome = c.getString(2);
         String categoria = c.getString(3);
         String email = c.getString(4);
         // código para capturar outros campos

         Presente presente = new Presente(id, descricao, nome, email, categoria, evento, data);
         presentes(presente);
      };
      return presentes;
   }
```

Consultando registros no banco de dados

Para recuperarmos os dados das colunas das tabelas, basta chamarmos o método **Cursor.get<Tipo>()**, informando o índice da coluna no banco de dados

```
@Override
   public List<Presente> obterTodos() {
      ArrayList<Presente> presentes = new ArrayList<Presente>();
      Cursor c = conexaoUtil.getConexaoSqlite().query(...);

      while (c.moveToNext()) {
         Integer id = c.getInt(0);
         String descricao = c.getString(1);
         String nome = c.getString(2);
         String categoria = c.getString(3);
         String email = c.getString(4);
         // código para capturar outros campos

         Presente presente = new Presente(id, descricao, nome, email, categoria, evento, data);
         presentes(presente);
      };
      return presentes;
   }
```

Consultando registros no banco de dados

```
@Override
   public List<Presente> obterTodos() {
      ArrayList<Presente> presentes = new ArrayList<Presente>();
      Cursor c = conexaoUtil.getConexaoSqlite().query(...);

      while (c.moveToNext()) {
         Integer id = c.getInt(c.getColumnIndex("id"));
         String descricao = c.getString(c.getColumnIndex("descricao"));
         String nome = c.getString(c.getColumnIndex("nome"));
         Integer categoria = c.getInt(c.getColumnIndex("categoria"));
         String email = c.getString(c.getColumnIndex("email"));
         // código para capturar outros campos

         Presente presente = new Presente(id, descricao, nome, email, categoria, evento, data);
         presentes(presente);
      };
      return presentes;
   }
```

Uma boa prática é **recuperar o índice** da coluna **dinamicamente** de acordo com o nome do campo

```
@Override
   public List<Presente> obterTodos() {
      ArrayList<Presente> presentes = new ArrayList<Presente>();
      Cursor c = conexaoUtil.getConexaoSqlite().query(...);

      while (c.moveToNext()) {
         Integer id = c.getInt(c.getColumnIndex("id"));
         String descricao = c.getString(c.getColumnIndex("descricao"));
         String nome = c.getString(c.getColumnIndex("nome"));
         Integer categoria = c.getInt(c.getColumnIndex("categoria"));
         String email = c.getString(c.getColumnIndex("email"));
         // código para capturar outros campos

         Presente presente = new Presente(id, descricao, nome, email, categoria, evento, data);
         presentes(presente);
      };
      return presentes;
   }
```

Inserindo registros no banco de dados

```
public void inserir(Presente p){
     ContentValues contentValues = new ContentValues();
     contentValues.put("nome", p.getNome());
     contentValues.put("email", p.getEmail());
     contentValues.put("categoria", p.getCategoria());
     conexaoUtil.getConexaoSqlite().insert("presentes", null, contentValues);
}
```

Inserindo registros no banco de dados

Para inserir, chamamos o método
SQLiteDatabase.insert()

```
public void inserir(Presente p){
     ContentValues contentValues = new ContentValues();
     contentValues.put("nome", p.getNome());
     contentValues.put("email", p.getEmail());
     contentValues.put("categoria", p.getCategoria());
     conexaoUtil.getConexaoSqlite().insert("presentes", null, contentValues);
}
```

Alterando registros no banco de dados

```
public void alterar(Presente p){
    ContentValues contentValues = new ContentValues();
      contentValues.put("nome", p.getNome());
      contentValues.put("email", p.getEmail());
      contentValues.put("categoria", p.getCategoria());
      conexaoUtil.getConexaoSqlite().update("presentes", contentValues,
                                            "id = " + p.getCodigo(), null);
}
```

Alterando registros no banco de dados

Para inserir, alterar, chamamos o método **SQLiteDatabase.update()**

```
public void alterar(Presente p){
    ContentValues contentValues = new ContentValues();
      contentValues.put("nome", p.getNome());
      contentValues.put("email", p.getEmail());
      contentValues.put("categoria", p.getCategoria());
      conexaoUtil.getConexaoSqlite().update("presentes", contentValues,
                                            "id = " + p.getCodigo(), null);
}
```

Excluindo registros no banco de dados

```
public void excluir(Presente p){
     conexaoUtil.getConexaoSqlite().delete("presentes", "id = ?",
                                   new String[] { Integer.toString( p.getId() ) } );
}
```

Excluindo registros no banco de dados

Para excluir, chamamos o método
SQLiteDatabase.delete()

```
public void excluir(Presente p){
      conexaoUtil.getConexaoSqlite().delete("presentes", "id = ?",
                                    new String[] { Integer.toString( p.getId() ) } );
}
```

Mão na Massa!

Mão na Massa!

1) Explore bastante os comandos SQL que podem ser utilizados na ferramenta sqlite3.

2) Crie uma app que importe um banco de dados já criado em algum front-end SQLite.

3) Crie uma app que permita inserir, consultar, alterar e deletar (CRUD) as informações do banco de dados.

Atenção: Utilizar a criação do banco de dados diretamente em código-fonte.