Criando um CRUD com Android Studio e SQLite

Veja nesse artigo como criar um CRUD em Android simples utilizando o Android Studio e o banco de dados nativo do Android, o SQLite.

As plataformas Mobile vêm ganhando força nos últimos anos, muitas empresas têm investido muito dinheiro para treinar e capacitar seus profissionais e torna-los capazes de **desenvolver aplicativos para dispositivos móveis**.

Neste artigo se mostrado como fazer um **CRUD** utilizando o sistema de banco de dados interno do Android, o **SQLite**. A criação desse banco de dados será feita diretamente com a SDK do Android e utilizaremos o Android Studio para a criação dos exemplos.

SQLite no Android

O SQLite é um banco de dados relacional open-source e fornece suporte para comandos SQL.

Cada **aplicação Android** pode criar quantos bancos de dados desejar e eles irão ficar armazenados no sistema, mas vale lembrar que o banco de dados criado pela aplicação não pode ser acessado nem visualizado por outra aplicação, apenas pela que o criou.

API de acesso

Para fazer o acesso ao banco de dados SQLite dentro da plataforma Android iremos utilizar uma API de acesso, a qual já vem no pacote SDK.

Duas classes serão utilizadas para a criação do banco de dados via aplicação e ambas podem ser vistas no código da **Listagem 1**:

1. **SQLiteDatabase**: Classe que contém os métodos de manipulação dos dados no banco;
2. **SQLiteOpenHelper**: Classe responsável pela criação do banco e também responsável pelo versionamento do mesmo.

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

```
    @Override
    public void onCreate(SQLiteDatabase db) {

    }

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

    }
}
```

Listagem 1. Criação de uma classe para criar o banco de dados

O código apresentado foi criado automaticamente pelo Android Studio e os dois métodos foram prototipados. Ao estender sua classe *SQLiteOpenHelper*, o **Android Studio** obriga o desenvolvedor a implementar dois métodos que são de suma importância para o correto funcionamento da criação do banco de dados:

- Método onCreate(): é chamado quando a aplicação cria o banco de dados pela primeira vez. Nesse método devem ter todas as diretrizes de criação e população inicial do banco.
- Método onUpgrade(): é o método responsável por atualizar o banco de dados com alguma informação estrutural que tenha sido alterada. Ele sempre é chamado quando uma atualização é necessária, para não ter nenhum tipo de inconsistência de dados entre o banco existente no aparelho e o novo que a aplicação irá utilizar.

Ambos recebem como parâmetro o objeto **db**, que é uma instância da classe SQLiteDatabase.

É notável também que o método onUpgrade(), além do objeto **db,** recebe dois outros parâmetros, que são inteiros: um contém a versão antiga da tabela e o outro contém a nova versão para ao qual o upgrade deve ser executado.

Agora criaremos uma table de cadastro de livros, com os seguintes campos: id, titulo, autor e editora.

O código da **Listagem 2** contém o nome do banco e a versão, que é um atributo importante e deve ser armazenado, pois com ela será possível efetuar alterações ao banco de dados sem problemas futuros, além do nome da tabela.

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

/**
 * Created by allanromanato on 5/27/15.
 */
public class CriaBanco extends SQLiteOpenHelper {
    private static final String NOME_BANCO = "banco.db";
    private static final String TABELA = "livros";
    private static final String ID = "_id";
    private static final String TITULO = "titulo";
    private static final String AUTOR = "autor";
    private static final String EDITORA = “editora";
    private static final int VERSAO = 1;

    @Override
    public void onCreate(SQLiteDatabase db) {

    }

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

    }
}
```

Listagem 2. Fragmento de código banco de dados

Na **Listagem 3** temos o código de criação do banco, que segue o mesmo padrão do JDBC, o qual permite que o desenvolvedor escreva comandos SQL em seu código e posteriormente chama um método que executará esse comando no banco, ou seja, o comando de criação da tabela deve ser escrito no código.

```
                        editora text
                        )
```

Listagem 3. Comando SQL para criação de banco de dados

Conheça os principais comandos SQL

Agora devemos pegar esse código e colocar dentro do método que irá criar o banco de dados, como mostra a **Listagem 4.**

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

/**
 * Created by allanromanato on 5/27/15.
 */
public class CriaBanco extends SQLiteOpenHelper {
    private static final String NOME_BANCO = "banco.db";
    private static final String TABELA = "livros";
    private static final String ID = "_id";
    private static final String TITULO = "titulo";
    private static final String AUTOR = "autor";
    private static final String EDITORA = "editora";
    private static final int VERSAO = 1;

    @Override
    public void onCreate(SQLiteDatabase db) {
        String sql = "CREATE TABLE"+TABELA+"("
                + ID + "integer primary key autoincrement,"
                + TITULO + "text,"
                + AUTOR + "text,"
                + EDITORA + "text"
                +")";
        db.execSQL(sql);
    }

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

    }
}
```

Listagem 4. Método onCreate() Implementado

Na **Listagem 5** temos o método onUpgrade(), que normalmente tem um comando SQL que apaga a tabela, se ela existir, e posteriormente invoca o método onCreate() para que recrie a tabela com as alterações feitas.

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

/**
 * Created by allanromanato on 5/27/15.
 */
public class CriaBanco extends SQLiteOpenHelper {
    private static final String NOME_BANCO = "banco.db";
    private static final String TABELA = "livros";
    private static final String ID = "_id";
    private static final String TITULO = "titulo";
    private static final String AUTOR = "autor";
    private static final String EDITORA = "editora";
    private static final int VERSAO = 1;

    @Override
    public void onCreate(SQLiteDatabase db) {
        String sql = "CREATE TABLE"+TABELA+"("
                + ID + "integer primary key autoincrement,"
                + TITULO + "text,"
                + AUTOR + "text,"
                + EDITORA + "text"
                +")";
        db.execSQL(sql);
    }

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

Na **Listagem 6** temos o construtor que passará para a super classe as informações do local e versão do banco.

```
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

/**
 * Created by allanromanato on 5/27/15.
 */
public class CriaBanco extends SQLiteOpenHelper {
    private static final String NOME_BANCO = "banco.db";
    private static final String TABELA = "livros";
    private static final String ID = "_id";
    private static final String TITULO = "titulo";
    private static final String AUTOR = "autor";
    private static final String EDITORA = "editora";
    private static final int VERSAO = 1;

    public CriaBanco(Context context){
        super(context, NOME_BANCO,null,VERSAO);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        String sql = "CREATE TABLE"+TABELA+"("
                + ID + "integer primary key autoincrement,"
                + TITULO + "text,"
                + AUTOR + "text,"
                + EDITORA + "text"
                +")";
        db.execSQL(sql);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS" + TABELA);
        onCreate(db);
    }
}
```

Listagem 6. Código funcional para criar banco de dados

Saiba mais sobre o Android Studio

CRUD - *Inserção de dados*

Na **Listagem 7** será criado um layout no Android Studio onde o usuário irá inserir as informações. O resultado pode ser conferido na **Figura 1**.

```
<RelativeLayout 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:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".Inicial"
    android:id="@+id/Inser">


    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Titulo:"
        android:id="@+id/textView"
        android:layout_marginTop="47dp"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
```

```
        android:id="@+id/editText"
        android:layout_below="@+id/textView"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Autor:"
        android:id="@+id/textView2"
        android:layout_below="@+id/editText"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="51dp" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText2"
        android:layout_below="@+id/textView2"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignRight="@+id/editText"
        android:layout_alignEnd="@+id/editText" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Editora:"
        android:id="@+id/textView3"
        android:layout_below="@+id/editText2"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="64dp" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText3"
        android:layout_below="@+id/textView3"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignRight="@+id/editText2"
        android:layout_alignEnd="@+id/editText2" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Cadastrar"
        android:id="@+id/button"
        android:layout_below="@+id/editText3"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="72dp"
        android:layout_alignRight="@+id/editText3"
        android:layout_alignEnd="@+id/editText3" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:id="@+id/textView4"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

</RelativeLayout>
```

Listagem 7. Código XML do layout do Insert

O código apresentado ainda não possui funcionalidade nenhuma, pois não foi programado dentro da Activity que o controlará.

Para facilitar o entendimento do código, ele será separado em duas classes: a classe que estende a Activity, responsável por controlar a UI, e a classe que será responsável por controlar as operações ao banco de dados.

A classe BancoController será responsável por controlar as manipulações ao banco, como mostra a **Listagem 8.**

```
public class BancoController {

    private SQLiteDatabase db;
    private CriaBanco banco;

    public BancoController(Context context){
        banco = new CriaBanco(context);
    }

    public String insereDado(String titulo, String autor, String editora){
        ContentValues valores;
        long resultado;

       db = banco.getWritableDatabase();
       valores = new ContentValues();
       valores.put(CriaBanco.TITULO, titulo);
       valores.put(CriaBanco.AUTOR, autor);
       valores.put(CriaBanco.EDITORA, editora);

       resultado = db.insert(CriaBanco.TABELA, null, valores);
       db.close();

       if (resultado ==-1)
           return "Erro ao inserir registro";
       else
           return "Registro Inserido com sucesso";

    }
}
```

Listagem 8. Código responsável por inserir dados

Veja que criamos um construtor público e instanciamos o atributo banco e o contexto que é passado por parâmetro a Activity.

É importante lembrar que o atributo **db** deve receber o resultado do método **getWritableDatabase**, que diz ao Android que o banco será utilizado para leitura e escrita de dados.

O método **insert** recebe como parâmetro a tabela em que os dados serão manipulados, um parâmetro nulo e o map com os dados que serão inseridos no banco no formato key/value. Além disso, temos a classe ContentValues para criar esse map. Lembre-se sempre de encerrar a conexão ao final de uma operação.

A **Listagem 9** mostra a Activity que controla a UI criada anteriormente e faz a chamada a esse método que insere o registro ao banco de dados.

```
public class InsereDado extends Activity {

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

        Button botao = (Button)findViewById(R.id.button);

        botao.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                BancoController crud = new BancoController(getBaseContext());
                EditText titulo = (EditText)findViewById(R.id.editText);
                EditText autor = (EditText)findViewById((R.id.editText2));
                EditText editora = (EditText)findViewById(R.id.editText3);
                String tituloString = titulo.getText().toString();
```

```
                Toast.makeText(getApplicationContext(), resultado, Toast.LENGTH_LONG).show();
            }
        });
    }
}
```

Listagem 9. Activity de Inserir Dado

Veja que será capturado o conteúdo dos EditTexts e convertido para String. Além disso, o método **insereDado** passa por parâmetro as informações a serem adicionadas no banco e, por fim, com o resultado retornado do método, será exibida uma mensagem (Toast) na tela mostrando se a operação foi um sucesso ou não.

CRUD - *Consulta aos dados*

Para realizar a consulta aos dados precisamos criar a interface, onde teremos um **ListView** e um arquivo de XML de layout para "estilizar" este, como mostra a **Listagem 10**. O resultado pode ser conferido na **Figura 2.**

```
<RelativeLayout 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:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context="br.com.home.bancodedados.Consulta">


    <ListView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/listView"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />
</RelativeLayout>
```

Listagem 10. Código da UI de consulta contendo ListView

Figura 2. ListView

Na **Listagem 11** temos o código do listview estilizado, onde o ID aparece do lado esquerdo da tela e o titulo do Livro do lado direito.

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnCount="2"
    android:rowCount="2" >


    <TextView
        android:id="@+id/idLivro"
        android:layout_width="0dip"
        android:layout_gravity="fill_horizontal"
        android:layout_height="wrap_content"
        android:layout_marginLeft="5dp"
        android:layout_marginTop="10dp"
        android:text="ID" />

    <TextView
        android:id="@+id/nomeLivro"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

Listagem 11. Continuação da UI

O GridLayout foi utilizado nesse exemplo apenas para mostrar os itens pesquisados separadamente na tela.

A **Listagem 12** mostra como é executada a operação de consulta de todos os dados no banco.

```
import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.content.Context;

/**
 * Created by allanromanato on 5/28/15.
 */
public class BancoController {

//Outros códigos…

public Cursor carregaDados(){
            Cursor cursor;
            String[] campos =  {banco.ID,banco.TITULO};
            db = banco.getReadableDatabase();
            cursor = db.query(banco.TABELA, campos, null, null, null, null, null, null);

            if(cursor!=null){
                cursor.moveToFirst();
            }
            db.close();
            return cursor;
    }
}
```

Listagem 12. Código de consulta ao Banco de Dados

É visível que o código para se carregar todos os dados também não utiliza nenhum comando SQL. Veja que são definidos os campos que a consulta retornará no array de Strings campo, depois o objeto **db** recebe o retorno do método **getReadableDatabase** que irá fazer com que os dados sejam acessados como somente para leitura. Após essa operação, o método **query** é chamado e o nome da tabela e os campos desejados para o retorno são passados por parâmetro, e esse método retorna um **Cursor**, uma classe do Android que salva as informações que são retornadas do banco de dados. Antes do cursor ser retornado para ser tratado na interface do usuário deve-se mover seu conteúdo para a primeira posição para que todos os dados sejam exibidos.

Agora precisamos tratar as informações exibidas para o usuário dentro do **ListView** estilizado que foi criado anteriormente. Uma novidade é a classe **SimpleCursorAdapter** que será utilizada como adaptador para que os dados contidos no cursor sejam devidamente exibidos na tela. A **Listagem 13** mostra como os dados serão colocados na tela utilizando essa classe.

```
import android.app.Activity;
import android.database.Cursor;
import android.os.Bundle;
import android.widget.AdapterView;
import android.widget.SimpleCursorAdapter;


public class Consulta extends Activity {
    private ListView lista;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_consulta);

        BancoController crud = new BancoController(getBaseContext());
        Cursor cursor = crud.carregaDados();

        String[] nomeCampos = new String[] {CriaBanco.ID, CriaBanco.TITULO};
        int[] idViews = new int[] {R.id.idLivro, R.id.nomeLivro};

        SimpleCursorAdapter adaptador = new SimpleCursorAdapter(getBaseContext(),
                R.layout.livros_layout,cursor,nomeCampos,idViews, 0);
```

Listagem 13. Código para mostrar os dados

O código irá instanciar a classe que faz o controle do banco de dados passando o contexto atual, que será carregado para dentro de um objeto do tipo **Cursor** com todos os dados recuperados do banco. Um array de Strings deve ser criado para armazenar os campos que deverão ser mostrados no **ListView** e juntamente um array de inteiros para armazenar o ID dos componentes que exibirão os dados.

No trecho onde o **SimpleCursorAdapter** é instanciado, vale uma atenção especial, pois é onde se “amarram" todas as informações: o layout onde está definido o estilo (nesse exemplo o GridLayout), o cursor que contém os dados, o array de Strings com o nome dos campos, o id dos componentes que serão utilizados para exibir o conteúdo e uma flag. Após concluído o processo, a lista deve ser amarrada com seu componente **ListView** e o adapter deve ser setado nessa lista. A **Figura 3** mostra a tela com as informações carregadas do banco de dados.

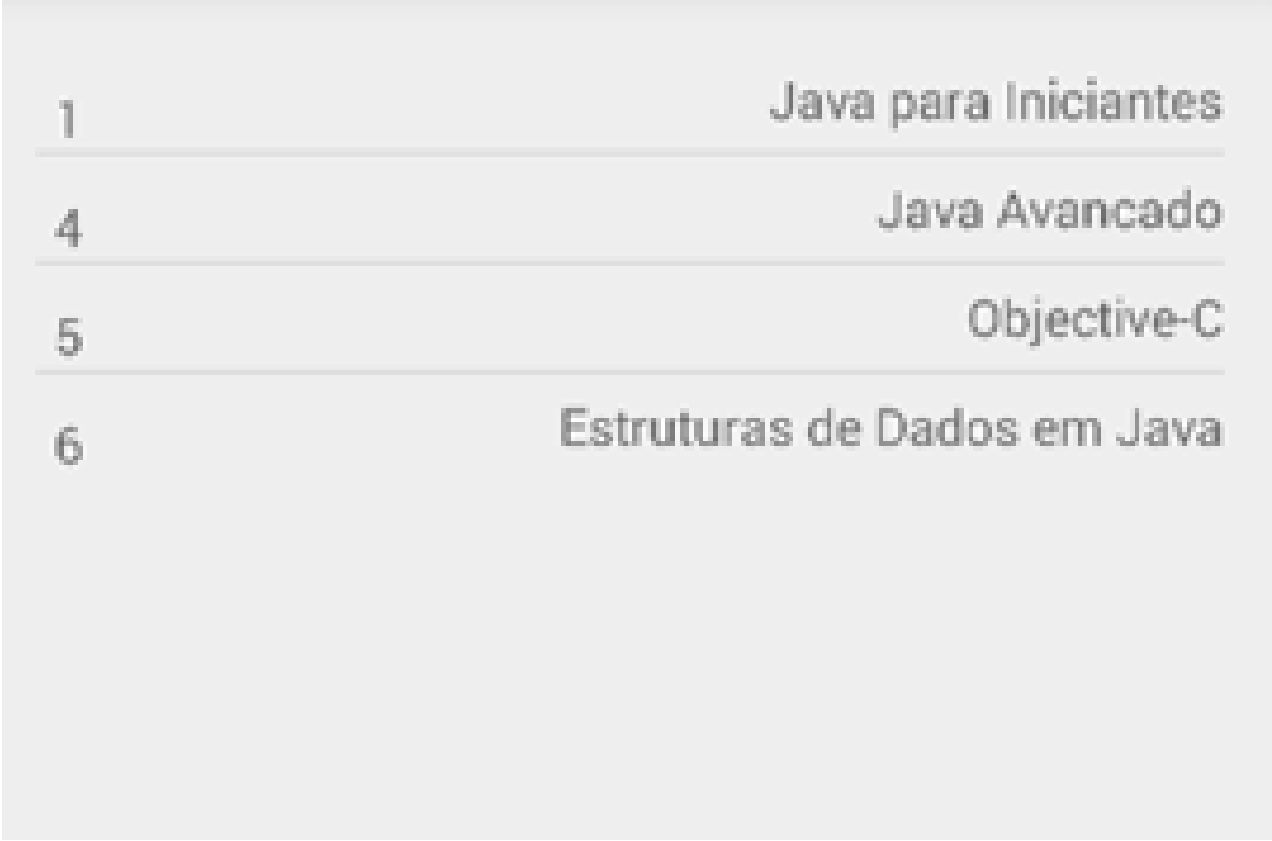

Figura 3. Resultado da execução

CRUD - *Alteração dos dados*

Para se alterar os dados contidos em um banco, devemos recuperar uma informação única de cada um deles e utilizá-la como uma referência para as demais. Primeiramente o código do layout será mostrado na **Listagem 14**. Percebe-se que o código é muito parecido com o layout de Inclusão de dados, mudando apenas alguns nomes de exibição.

Na **Figura 4** podemos conferir o resultado.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
```

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Titulo:"
        android:id="@+id/textView4"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="75dp" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText4"
        android:layout_below="@+id/textView4"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Autor:"
        android:id="@+id/textView5"
        android:layout_below="@+id/editText4"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="45dp" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText5"
        android:layout_below="@+id/textView5"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignRight="@+id/editText4"
        android:layout_alignEnd="@+id/editText4" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Editora:"
        android:id="@+id/textView6"
        android:layout_below="@+id/editText5"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="39dp" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText6"
        android:layout_below="@+id/textView6"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Alterar"
        android:id="@+id/button2"
        android:layout_below="@+id/editText6"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="51dp"
        android:layout_alignRight="@+id/editText6"
        android:layout_alignEnd="@+id/editText6" />
</RelativeLayout>
```

Listagem 14. Layout da Alteração

Figura 4. Layout de Alteração

Existem várias formas de se chegar nessa tela, mas a adotada nesse artigo é o clique na informação desejada no **ListView** de consulta, onde direcionará todas as informações para essa tela. A **Listagem 15** mostra apenas o código que deve ser adicionado ao

```
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                String codigo;
                cursor.moveToPosition(position);
                codigo = cursor.getString(cursor.getColumnIndexOrThrow(CriaBanco.ID));
                Intent intent = new Intent(Consulta.this, Alterar.class);
                intent.putExtra("codigo", codigo);
                startActivity(intent);
                finish();
            }
        });
```

Listagem 15. Complemento a **Listagem 13**

Ao ser clicado, o conteúdo do cursor para a primeira posição captura o código que está contido dentro do cursor referenciado por **CriaBanco.ID**, e esse código é passado via **Intent** para a activity que será estudada agora.

Após a passagem do código de referência das informações contidas no banco de dados, é necessário que seja executada uma consulta ao banco e a informação seja recuperada e armazenada em um objeto do tipo **Cursor**. A **Listagem 16** mostra como isso será feito.

```
import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.content.Context;

/**
 * Created by allanromanato on 5/28/15.
 */
public class BancoController {

            //Outros códigos…
public Cursor carregaDadoById(int id){
        Cursor cursor;
        String[] campos =  {banco.ID,banco.TITULO,banco.AUTOR,banco.EDITORA};
        String where = CriaBanco.ID + "=" + id;
        db = banco.getReadableDatabase();
        cursor = db.query(CriaBanco.TABELA,campos,where, null, null, null, null, null);

        if(cursor!=null){
            cursor.moveToFirst();
        }
        db.close();
        return cursor;
    }
}
```

Listagem 16. Código para carregar informação especifica

Aqui pode se notar uma diferença em relação ao código de recuperação de dados, pois aqui são recuperados todos os campos dentro do array de Strings. Um atributo do tipo String é declarado para armazenar a cláusula **WHERE,** que diz ao banco de dados para devolver apenas registros com o determinado id passado por parâmetro. A query é executada passando agora como parâmetro a condição para que as informações sejam recuperadas.

A **Listagem 17** mostra claramente como a exibição na tela acontecerá.

```
import android.app.Activity;
import android.content.Intent;
import android.database.Cursor;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;


public class Alterar extends Activity {
    EditText livro;
    EditText autor;
    EditText editora;
```

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

        codigo = this.getIntent().getStringExtra("codigo");

        crud = new BancoController(getBaseContext());

        livro = (EditText)findViewById(R.id.editText4);
        autor = (EditText)findViewById(R.id.editText5);
        editora = (EditText)findViewById(R.id.editText6);

        alterar = (Button)findViewById(R.id.button2);

        cursor = crud.carregaDadoById(Integer.parseInt(codigo));
        livro.setText(cursor.getString(cursor.getColumnIndexOrThrow(CriaBanco.TITULO)));
        autor.setText(cursor.getString(cursor.getColumnIndexOrThrow(CriaBanco.AUTOR)));
        editora.setText(cursor.getString(cursor.getColumnIndexOrThrow(CriaBanco.EDITORA)));

    }
}
```

Listagem 17. Exibir dado na tela

Veja que é recuperado o código passado pela **Intent** que chamou e instancia o objeto responsável por manipular o banco de dados, iniciando os objetos de tela. Logo após ele irá carregar os dados utilizando o método do controle **carregaDadosById** passando o código como parâmetro. Os componentes de tela (livro, autor e editora) recebem o dado a partir da leitura das informações contidas no cursor.

Para os dados serem alterados precisamos implementar o método **alteraRegisto** no controller do banco de dados, como mostra a **Listagem 18.**

```
import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.content.Context;

/**
 * Created by allanromanato on 5/28/15.
 */
public class BancoController {

            //Outros códigos…
   public void alteraRegistro(int id, String titulo, String autor, String editora){
        ContentValues valores;
        String where;

        db = banco.getWritableDatabase();

        where = CriaBanco.ID + "=" + id;

        valores = new ContentValues();
        valores.put(CriaBanco.TITULO, titulo);
        valores.put(CriaBanco.AUTOR, autor);
        valores.put(CriaBanco.EDITORA, editora);

        db.update(CriaBanco.TABELA,valores,where,null);
        db.close();
    }
}
```

Listagem 18. Altera Registro no Banco de dados

Na **Listagem 19** temos o comando para chamar o método update.

```
alterar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                crud.alteraRegistro(Integer.parseInt(codigo), livro.getText().toString(),autor.getText().toString(),
                        editora.getText().toString());
```

Listagem 19. Chamar o método update

Esse código é um evento que chamará o método **alteraRegistro** e, logo após, chamará a Activity responsável por exibir os dados na tela, confirmando assim se a informação foi alterada ou não.

CRUD - *Deletando dados*

Por conveniência, o deletar será utilizado juntamente com o alterar, na mesma Activity, portanto para o componente de layout, insira o seguinte código mostrado na **Listagem 20** após o término do código da **Listagem 14.**

```
<Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Deletar"
        android:id="@+id/button3"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignRight="@+id/button2"
        android:layout_alignEnd="@+id/button2" />
```

Listagem 20. Adicionando o botão Deletar

A **Figura 5** mostra como o layout deverá ficar após inserir o código XML.

A lógica utilizada para carregar os dados que serão deletados pelo usuário é a mesma que a utilizada pelo alterar, por estarem na mesma activity.

A **Listagem 21** mostra o método da classe de controle do banco responsável por deletar o dado selecionado.

```
public void deletaRegistro(int id){
        String where = CriaBanco.ID + "=" + id;
        db = banco.getReadableDatabase();
        db.delete(CriaBanco.TABELA,where,null);
        db.close();
    }
```

Listagem 21. Código que deleta os dados

Veja que os dados do banco serão apenas apagados e não manipulados. O método **delete** receberá o nome da tabela e a cláusula where.

A Activity que controla a tela deve receber as diretivas responsáveis por acionar o botão e chamar o método de deleção do dado. Referindo-se a **Listagem 18** deve-adicionar algumas linhas de código para que ela se adeque ao esperado. A **Listagem 22** mostra os códigos necessários para completar.

```
//Código Antes
deletar = (Button)findViewById(R.id.button3);
deletar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                crud.deletaRegistro(Integer.parseInt(codigo));
                Intent intent = new Intent(Alterar.this,Consulta.class);
                startActivity(intent);
                finish();
            }
        });
```

Listagem 22. Funcionalidade Delete

É muito parecido com o alterar, mas aqui o **deletaRegistro** é chamado ao invés do **alteraRegistro.** Da mesma forma, a activity que mostra os dados na tela é chamada para confirmar a deleção.

Com isso, o CRUD está concluído, mas é claro que esses códigos podem ser melhorados, adaptados e modificados segundo as necessidades do projeto.

Espero que tenho gostado desse artigo e até a próxima oportunidade.

Aprenda mais sobre CRUD com SQLite no Android

Links

Android Studio

http://developer.android.com/tools/studio/index.html

SQLite

http://www.sqliteexpert.com/download.html

Curso relacionado: Criando uma loja virtual com Android Studio